3D Reconfigurable Autopilot Flight System for both General Aviation and Unmanned Aerial Systems

by

Pedro Pablo Plazas Rincon

Submitted in fulfilment of the requirements for the degree of

Master of Engineering by Research

Science and Engineering Faculty

Queensland University of Technology

March 2015

Keywords

Autopilot Flight Systems, CAN protocol, bus interfaces, open-source projects, servos, hardware architecture, safety standards, PID Controller, Unmanned Aerial Systems (UAS), General Aviation, microcontroller.

Abstract

Autopilot Flight Systems (AFS) are essential to guide manned or unmanned aircraft without human assistance during long flights. AFS have commonly been utilised in the General Aviation (GA) that specialises in light aircraft used for civilian purposes like business, training and recreational flights. The AFS allow GA pilots to focus on other tasks while AFS control the aircraft, reducing the workload and increasing airborne safety. At the same time, Unmanned Aerial Systems (UAS) require an AFS to guide the unmanned aircraft to follow a pre-defined trajectory or to command the platform to perform manoeuvres in case of an on-board emergency or loss of communication with the Ground Station (GS).

GA Aircraft and UAS AFS have the same aerodynamic foundations and offer similar flight modes, however no commercial or research project has used a 3D AFS in both GA Aircraft and Fixed-Wing UAS. This research presents the design, development and implementation of a Reconfigurable Autopilot Flight System (RAFS) for both a GA Aircraft and Fixed-Wing UAS. Modules that interconnect UAS autopilot with a group of servos from a UAS and a Cessna Aircraft were developed using the CAN protocol. That is, a CAN driver was implemented for UAS autopilot, was developed software for a Bridge module and the Cessna Servos were adapted using two H-Bridge modules. A standard PID control position was also developed to control the position of Cessna Servos.

RAFS was developed using open source software and hardware and results showed that an UAS AFS can be reconfigured to work with a GA Aircraft using a modular architecture to work for both types of aircraft. This opens possibilities to develop avionics modules and applications equally for manned and unmanned aircraft. For instance, if an AFS can be re-configured to work on a UAS or GA Aircraft, the AFS can be tested on an UAS first before it is transitioned to the final GA Aircraft. This procedure decreases costs for testing new avionics equipment and reduces risks for test pilots who must evaluate the new development for manned aircraft.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ADC	Analogue Digital Converter
ADAHRS	Air Data Attitude Heading System
AFDC	Autopilot Flight Director Computers
AFDS	Autopilot Flight Director System
AFS	Autopilot Flight Systems
AHRS	Attitude and Heading Reference System
ARINC	Aeronautical Radio Incorporated
ATC	Air Traffic Control
BCA	Back-drive Control Actuator
CAN	Control Area Network
CORBA	Common Object Request Broken Architecture
COTS	Commercial Off-the-Shelf
CS	Chip Select
DSC	Digital Signal Controller
DSP	Digital Signal Processor
EI	Electromagnetic Interference
EMIF	External Memory Interface
ESA	European Space Agency
EUROCAE	European Organisation for Civil Aviation Equipment
FAA	Federal Aviation Administration
FCSG	Flight Control System Gateway
FPGA	Field Programmable Gate Array
FPU	Flight Processing Units
GA	General Aviation
GPL	General Public License
GPS	Global Position System
GRDC	Gulf Range Drone Control
HMI	Human Machine Interface
ICR	Input Capture Register
I^2C	Inter-Integrated Electronics Circuit
IAU	Integrated Avionics Units
MOPS	Minimum Operation Performance Standards
MISRA	Motor Industry Software Reliability Association
MCC	Main Control Computer
MCP	Mode Control Panel
MUAV	Mini Unmanned Aerial Vehicles

NAS	National Aerial Space
NU	Navigation Unit
OCR	Output Compare Register
OOP	Object Oriented Programming
PID	Proportional Integral Derivative
PFD	Primary Flight Display
POSIX	Portable Operating System Interface
PWM	Pulse-Width Modulation
RAFS	Reconfigurable Autopilot Flight System
RAMA	Remotely Operated Aerial Model Autopilot
RAVEN	Reconfigurable Autopilot for Vehicles with Enhanced Navigation
RTCA	Radio Technical Commission for Aeronautics
RTOS	Real Time Operating System
SCU	Servo Control Unit
SDI	Serial Data Input
SUAV	Small Unmanned Aerial Vehicle
SPI	Serial Peripheral Interface
SOA	Service Oriented Architecture
UART	Universal Asynchronous Receiver/Transmitter
UAS	Unmanned Aerial Systems
UAV	Unmanned Aerial Vehicle

Statement of Original Authorship

The work contained in this thesis has not been previously submitted to meet requirements for any award at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

QUT Verified Signature

Signature:

Date: 24/03/2015

Acknowledgements

I would like to thank my principal supervisor Dr. Luis Mejias Alvarez for sharing his knowledge, supporting and guiding me during this research and at the same time to thank ARCAA Director Dr. Duncan Campbell who was the associate supervisor during this research project.

This research had not been possible without the cooperation of ARCAA staff specially Engineer Duncan Greer who contributed with his experience and knowledge as Engineer and pilot. I want to thank the Engineering Laboratory Staff from S Block level 9 for their excellent service with the electronics equipment. My sincere gratitude to Dr Christian Long and Ms Karyn Gonano of the QUT Academic Language and Learning Service who support the edition and correction of this monograph. Also, a special thanks to Library Staff in both campuses Kelvin Grove and Gardens Point for their service during this time. I want to mention to my fellows Master and PhD Research students in both in ARCAA and Engineering Faculty for sharing me their passion and knowledge in each one of their research projects.

Finally, I wish to dedicate this investigation to my mother Ana, my father Pablo, my brother and my sisters who support me a lot of during all this time. Also, I want to thank to Ximena for sending me her best positive feelings and thoughts throughout the most difficult moments. This effort is for all them and all “Pablo Pueblo” that walk by the streets of Latin-American Countries. Millions of thanks QUT!

Chapter 1: Introduction

Aircraft complexity combined with long flights mean Autopilot Flight Systems (AFS) play an essential role in General Aviation. These AFS are very useful because they command and guide the aircraft without human assistance, especially when flights are routine and monotonous. As a consequence, pilot workload is reduced, allowing them to concentrate on other vital flight tasks, while the autopilot steers the aircraft. Also, pilot fatigue and stress are decreased during long trips, reducing human failures and increasing aerial safety.

AFS are not only used in piloted aircraft but are also commonly used in Unmanned Aerial Systems (UAS). UAS have become more sophisticated technologically increasing their capabilities to conduct complex tasks particularly in dangerous situations where UAS can replace a piloted aircraft to avoid putting the crew at risk. One of the main advantages of using UAS is that they can be remotely piloted by an operator from a ground station. Similar to aircraft pilots, UAS operators need to use autopilots when UAS missions may be complex or extend over many hours.

For both piloted civil aircraft and UAS the AFS have the same basic functions, known as autopilot modes. These autopilot modes allow an aircraft to keep a desired altitude, steer automatically in one direction, follow a new trajectory using GPS, and keep a desired airspeed or a constant climb-rate. Despite having similar flight modes, software and hardware engineers have developed specialised AFS for each specific platform either manned or unmanned aircraft owing to the fact that each type of aircraft has different safety criteria and testing methodologies.

The convergence between civil aviation and UAS will be evident not only by sharing the same Airspace [1] but also using the same avionics hardware and software systems. As a result, AFS equipment could be implemented and installed in any type of UAS or GA Aircraft without requiring major modifications. For instance, an AFS installed in GA Aircraft model could be transferred and installed on a UAS or vice versa.

A first step in this convergence process was on September 2013 when the Boeing Company modified an old piloted aircraft F-16 and transformed this aircraft in an unmanned aircraft model QF-16 controlled from a ground control station [2]. Boeing engineers adapted an F-16 flight control computer into a system known as Gulf Range Drone Control (GRDC) a Boeing platform used to control UAS from a ground station. The autopilot control system was reconfigured to run on the QF-16 hardware to send and receive commands and send and receive telemetry, and visuals [3]. A similar

project to adapt an UAS platform in a military manned aircraft was purposed by the Italian Company Alenia Aermacchia which conceived a project to convert F-104s to UAS [4]. Both projects worked for military applications only using avionics components for manned aircraft, and did not include UAS components.

The development of an AFS for both manned and unmanned aviation poses many challenges. One is related to the integration of actuators for GA aircraft and Fixed-Wing UAS. Another is the implementation of protocol interfaces used by avionics manufacturers which can be adapted by UAS as well. The design must also be modular enough for the autopilot to integrate the GA aircraft and UAS modules when configured to work with one or another type of aircraft. The solution would be to design and implement a RAFS based on an open-source UAS autopilot and then to develop of interfacing modules to integrate avionics components for both GA aircraft and Fixed-Wing UAS.

1.1 RESEARCH PROBLEM

For GAA and UAS both types of aircraft have a similar group of actuators and sensors that interact with the AFS. The actuators are used to move the flight control surfaces (ailerons, elevator, and rudder) and the set of sensors used to read the physical variables (altitude, airspeed, vertical speed, GPS position, and attitude) necessary to know the flight conditions. These physical variables are displayed in the control panel (machine interfaces) on-board of GAA cockpit or are sent to ground station when the aircraft is a UAS. Also, the communication system in GAA and UAS allows transmitting and receiving data between an aircraft and a ground station to check the aircraft condition. At the same time, the path planning module is used to program the trajectory that the AFS must follow according to the coordinates supplied by the GA pilots or UAS operators.

Taking advantage that some of these modules have the same functionalities, the main objective of this research is to design, develop and implement a 3D Reconfigurable Autopilot Flight System (RAFS) which could be used for both a GA aircraft and a Fixed-Wing UAS (Figure 1.1). In order to achieve this objective, hardware and software architecture will be purposed and implemented to adapt an UAS Autopilot Flight System to work with fixed-wing unmanned and manned aircraft. This research will focus on how to interface servos which control the aircraft primary flight control surfaces with a UAS autopilot. The following questions will guide this research.

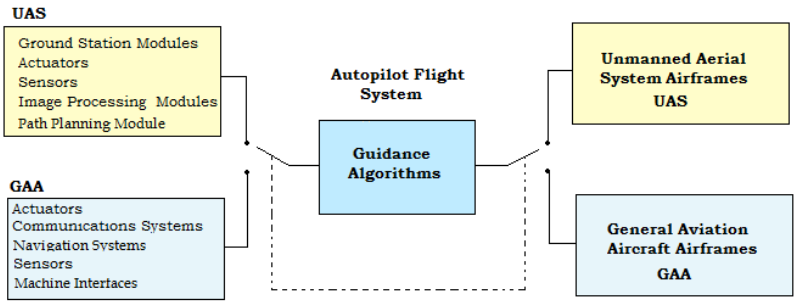


*Figure 1:1*Autopilot Flight System for Both Fixed-Wing UAS and GA Aircraft

- **Question 1:** *How can an Autopilot Flight System be designed to work for both wing-fixed piloted aircraft and fixed-wing UAS?*

  The differences and similarities between AFS for manned and unmanned aircraft will be examined by analysing the features of autopilot modes, size, weight, communication systems and power supply. For example, an AFS used in a GA aircraft would not be compatible for an UAS due to its on-board payload size and power constraints. Servo motors are completely different in terms of power, voltage and control for a piloted aircraft and a UAS. Commercial piloted aircraft autopilots do not offer communication with a ground station, essential to control a UAS. Therefore, this research assesses the possibility of implement a UAS AFS on board a GA Aircraft due to its advantages of size, weight and reusability of hardware and software.

- **Question 2:** *What software and hardware criteria shall be implemented in the design of a Reconfigurable Autopilot Flight System?*

  Internal access to commercial Autopilot Systems for both UAS and GA Aircraft is restricted due to confidentiality making open-source hardware and software UAS autopilots an excellent alternative. These systems allow implementing and adapting new components because it is possible to study their hardware and software without licensing restrictions or copyright. Some open source AFS have a highly modular design which offer a number of interfaces to communicate with others systems, which means they may connect GA Aircraft modules.

1.2 CONTRIBUTION OF RESEARCH

The major contribution of this research is the development of hardware and software architecture to implement a Reconfigurable Autopilot Flight System (RAFS) for use in both UAS and GA aircraft only focusing on the actuators. The derived contributions of this research are:

- RAFS design based on open-source hardware and software which allows cost reduction when compared with a Commercial off-the-shelf (COTS) autopilot for GA Aircraft. This research demonstrated that AFS architecture based on open-source hardware and software can be used for both GA Aircraft and UAS.
- An open hardware and software Bridge module was developed to isolate and integrate the group of servos for both a manned aircraft and unmanned aircraft to the AFS. This is critical since the Bridge module allowed integrating and configuring modules of the RAFS without requiring major software and hardware modifications.
- A CAN interface was developed to communicate the Autopilot with the Bridge module which ensured a reliable data transmission for both hardware modules. This research demonstrated that the use of CAN protocol is a better alternative to communicate avionics modules on a UAS AFS instead of low level protocols such as I^2C or SPI. CAN protocol is one of the standards used in the avionics industry and the use of this interface allows integration with other GA aircraft modules.
- Software pre-processor directives were developed to implement the RAFS configuration to prevent logic failures for unused codes. These directives deactivate in execution time the used code for an UAS while RAFS works for a GA aircraft and vice versa.
- The RAFS design provides a way to transform a GA Aircraft into a UAS by just re-configuring the ground station modules to transmit and send data to GAA. It means that GAA could be piloted remotely by an operator from a ground station. For example, in emergency situations pilots were unable to pilot the aircraft due to sudden health problems.

1.3 THESIS OUTLINE

This thesis is structured as follows;

Chapter 2 shows a review of the different protocols used in the Avionics industry and UAS to interface different modules using low and high level protocols. This chapter review also several AFS for UAS and GA aircraft used in commercial and research projects analysing their hardware architectures. In the last part of Chapter 2 some open-source AFS are reviewed, which bring the possibility to implement new hardware and software applications.

Chapter 3 outlines the proposed architecture explaining each module and the connections to transmit and receive data. In addition, the CAN Network implementation is analysed indicating each of the CAN node implemented. Finally, a software and hardware description is presented describing the main features for each element.

Chapter 4 describes the RAFS Software implementation detailing the software components developed or modified to connect the modules.

Chapter 5 presents the results and data obtained during this research showing the transmitted CAN messages through different nodes, the PWM signals generated to change the UAS servos positions. Also, the results of PID controller for Pitch and Roll Cessna Servos and the analysed data for each servo are analysed and shown.

Chapter 6 provides the conclusions and future work that can be developed based on this research. Following this Chapter, Appendix A details the Avionics standards suggested to develop an Autopilot Flight System, the Appendix B shows main basic related to CAN protocol like physical and logic features and the Appendix C shows the operation of an H-Bridge. Appendix D shows the components of RAFS, Appendixes E and F show respectively the schematics for Pixhawk Autopilot I/O and AVR-CAN board. Finally, the Appendix G shows a wiring diagram and logic commands to control the H-Bridge MC33926.

Chapter 2: Literature Review

The design, configuration and modularity of an AFS for UAS and General Aviation (GA) aircraft has two main functional blocks, one is the control, navigation and guidance algorithms, and the second is the on-board computer architecture and its connectivity with peripherals. The first block is related to autopilot flight modes in an aircraft and how these algorithms can guide different manned and unmanned aircraft in autonomous operation. This approach does not form part of this research because this project bases the development on an open-source AFS which has implemented these algorithms previously. Furthermore, the design of new navigation algorithms would imply a long process in time which is beyond the scope of this research. The block addresses two components; the first is how the autopilot hardware architecture provides different bus interfaces to connect the set of sensors, actuators and future modules with an AFS. The second component is how the hardware design provides modularity to execute the autopilot algorithms in more specialized microprocessors and how the peripherals are interfaced to autopilot. These approaches are considered when using an AFS model with different aircraft or making an AFS compatible with different sensors or actuators.

This chapter presents an overview of the interfaces and protocols used for GA and UAS in Section 2.1. The Section 2.2 analyses civil and commercial aircraft autopilots emphasising the connectivity between actuators and distinguishing the type of bus interfaces used to integrate the avionics modules. Technical details about the type of hardware devices used are not provided by manufacturers. The Section 2.3 reviews the hardware architecture commercial and open-source autopilots. Depending on the type of microprocessor or bus interface are classified the UAS autopilot architectures in those based in microcontrollers, Digital Signal Controllers (DSCs), Digital Signal Processors (DSPs) or Field Programmable Gate Array (FPGA). Later, UAS autopilot architectures base on Control Area Network (CAN) protocol and architectures based on distributed designs such Service Oriented Architecture (SOA) and Common Object Request Broken Architecture (CORBA) are analysed.

2.1 BUS INTERFACES USED FOR GENERAL AVIATION AND UAS AUTOPILOTS

The aerospace industry has developed standards for digital data transmission such as ARINC 429 [5], ARINC 629 [6] and the CAN-based protocols [7] ARINC 812, ARINC 825 [8], ARINC 826, CANopen [9], and CANAerospace [10]. The implementation of serial data bus systems for civil aviation was introduced in 1977 using the protocol ARINC 429. Since the 80's, the commercial aircraft Boeing 757, 767 and the Airbus A310 used ARINC 429 to integrate their avionics systems reducing cabling, weight, and power consumption increasing modularity at the same time [11]. However, new civil aircraft require high data transfer and ARINC 429 only supports a maximum speed of 100kbits with only one node transmitting data at any time. Therefore, if duplex communication is a need, it will require an additional link which increases the number of interconnections [12].

In 1986, Boeing released the protocol ARINC 629 which improved the features of ARINC429 increasing the flexibility and which could be update for recent versions. This protocol was designed to support short-circuit failures and increase the immunity against Electromagnetic Interference (EI) because of the use of twisted pair wires. The maximum bus length to connect components is 100 meters and the data rate of 2 MHz which much higher than ARINC 429.

After being successful on the automotive industry, CAN protocol was adapted by aircraft manufacturers to design the connectivity between different avionics modules. Three main advantages CAN protocol provides to airborne applications: one is the high EI immunity, second is the excellent error detection and finally, the high flexibility to add multiples nodes into a CAN Network. All these advantages increase the safety conditions to process data in General and Commercial Aviation. **Appendix B** details additional features of the CAN protocol such as description of layers, frames and data bus.

However, CAN protocol do not provide a high level of data transfer because it has only a physical and data link layers. As consequence, the airborne industry developed a series of CAN-based protocols which have added new layers to manage and transport the CAN messages. For instance, ARINC 812 is developed to interoperate between different aircraft types focusing on the mechanical and electrical features and the power management [7]. Other CAN-based protocol is ARINC 825 which is similar to ARINC 812, but ARINC 825 includes properties to interconnect different CAN sub-systems and brings tools to address and communicate the different nodes inside the CAN networks. ARINC 825 defines structures to interpret the data received for each node and provides emergency event mechanism to identify problems inside the network [13].

CANopen protocol [9] is other alternative used by avionics manufacturers to integrate modules on-board to aircraft. Similar to ARIN 825, CANopen defines the mechanism to communicate the different nodes on a CAN Network, including predetermined data structures used to create the CAN messages. CANopen brings a high level of flexibility but its implementation is complex and requires high efforts to assure the safety standards. Similar to CANopen, the CANAerospace protocol is open and it allows avionics manufacturers develop modular components increasing the interoperability. CANAerospace provides mechanisms to establish the communication between the different nodes on the CAN network. These mechanisms include messages synchronization, failure detection, block data transfer and categorization of messages according to the aircraft systems [10].

Although the UAS manufacturers have not adapted the same bus standards used in General Aviation. UAS autopilots use a set of low-level protocols such as RS232, I^2C and SPI which communicate the autopilot with sensors and servos. However, the number of devices that can be connected with an autopilot is limited and it depends on the microcomputer architecture. These protocols do not bring further mechanisms to control the messages sent and received by the different components connected to Network. In addition, error detection, EM immunity and the bus length is limited compared with GA protocols. However, some UAS Autopilot models have included the CAN interface which allows to UAS work with more advanced avionics systems increasing its modularity.

The RS232 interface standard has been commonly used to communicate microprocessors with peripherals; RS232 is a serial communication approach that can transmit messages synchronously or asynchronously in one (simplex) or two directions (duplex) using a data transfer rate between 50 and 115200 baud. The maximum cable length using a baud rate of 110 is 850m whereas using a baud rate of 19200 baud the length decreases considerably to 50 meters. This standard only works in point-to-point connection which means that only two devices can communicate at the same time. However, RS232 can be used to implement daisy-chain network where more than two terminals can be connected, but this increases the risk of a failure communication [14]. This interface is used in some GA autopilot models to connect servos and sensors as well.

The Serial Peripheral Interface (SPI) bus is based on master-slave synchronous communication where the master device controls and selects the communication with different slave components (Figure 2.1). The SPI uses four signals to establish the communication: Serial Clock (CLK), Serial Data Output (SDO), Serial Data Input (SDI) and Chip Select (CS). SPI protocol does not define a determined rate of transmission but it is possible to get speed up of 8.75 MB/s [15].

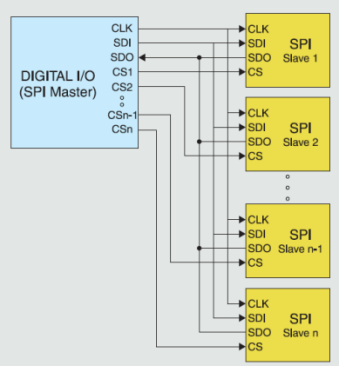


Figure 2:1 SPI Wiring Connection Master-Slave Configuration [15, p.12].

The Inter-Integrated Electronics Circuit (I^2C) [15] is a serial protocol which uses two lines one for data transmission (SDA – Serial Data) and other for clock signal (SLC – Serial Clock). I^2C uses the multi-master mechanism where peripherals are interconnected as masters or slaves on the network (Figure 2.2). The rate of transmission can be configured between 100Kb/s, 400 Kb/s and 3.4 Mb/s.

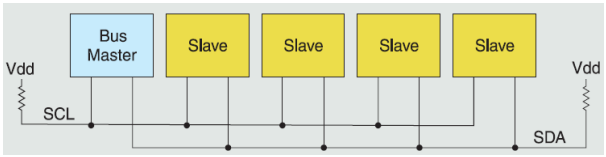
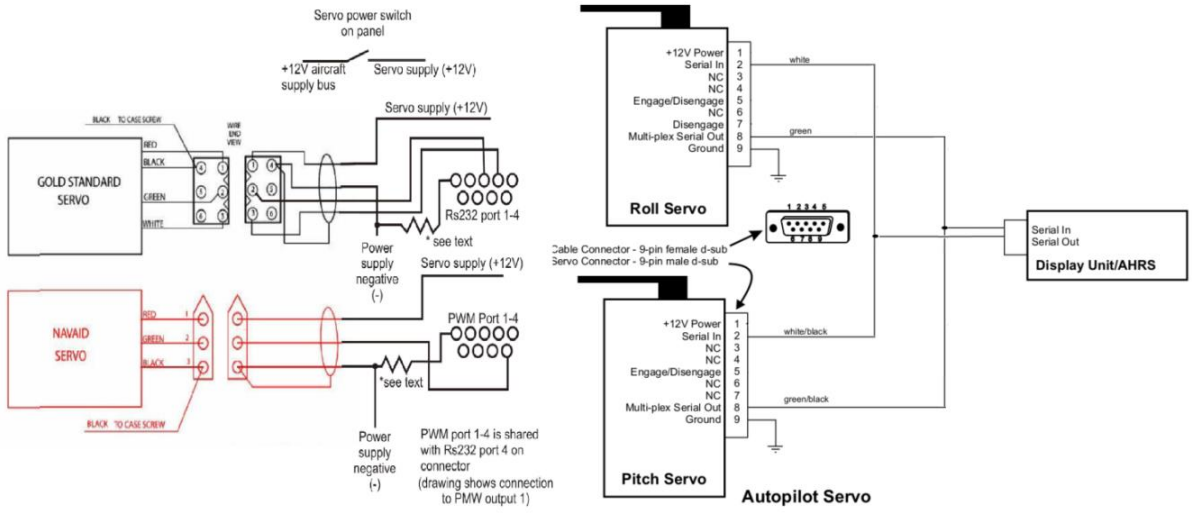


Figure 2:2 I^2C Wiring Connection [15, p.9]

2.2 GENERAL AVIATION AUTOPILOT ARCHITECTURE

Avionics product market develops generic hardware and software modules which mean that can be installed for different civil aircraft. These generic airborne products included AFS are known as Commercial Off-the-Shelf (COTS) [16]. These Avionics Architectures become open systems where aircraft applications can be shared and used by many on-board modules including Display, Communication, Navigation, Recording, Radar and Engine Systems [17]. This Section presents the civil autopilot designs based on servos or sensors configuration and bus interfaces used.

The company MGL Avionics uses for their Autopilot models Xtreme, Enigma and Odyssey [18] the interface RS232 to communicate their set of servos. Each servo requires a RS232 port to transmit its current position and status data and receive the desired position from autopilot. Changing the autopilot configuration using the same port RS232 can be used to connect PWM servos. A similar autopilot servo wiring using the RS232 interface was designed by GRT Autopilot [19], but Pitch and Roll Servos use the same autopilot port. GRT Autopilot has the advantage in that it does not require a RS232 port per servo whereas the MGL Avionics model requires a RS232 per each servo, as shown in Figure 2.3.



(a) MGL Avionics Autopilot [18, p.33]. (b) GRT Autopilot [19, p.16].

Figure 2:3 Servo Wiring Configuration for GA aircraft AFS models

Exchanging servos between MGL and GRT autopilot models can be difficult because there is no compatibility between servos due to each manufacturer has adapted the interface RS232 with different designs. The installation manual and electrical diagrams show that each servo has a hardware and software interface that interprets the commands sent by its autopilot. Whereas the MGL Autopilot implements an RS232 or PWM interface per servo, the GRT Autopilot has a common RS232 interface shared for both Roll and Pitch Servos.

The company Trutrak Flight Systems in its model EFIS AP101 Autopilot [20] presents an intermediate design with the possibility to include new avionics elements using the interfaces RS232 and ARINC 429. For example, these ports integrate different GPS Garmin series or King KMD. However, the outputs to roll and pitch servos are connected directly to autopilot with four control lines used to move the servo in the desired sense and speed. The servo units are not modular because these are customized for specific manufacturer servo models where the roll and pitch servos have a particular number of lines of connection. This lack of compatibility occurs with the pitot tube and static sensors which do not use a bus interface to transmit the measurements (Figure 2.4).

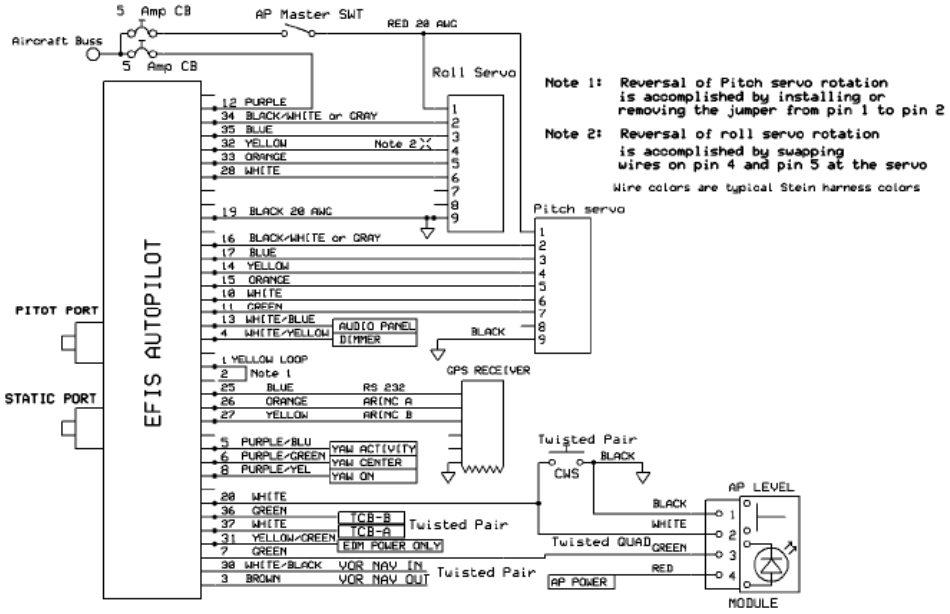
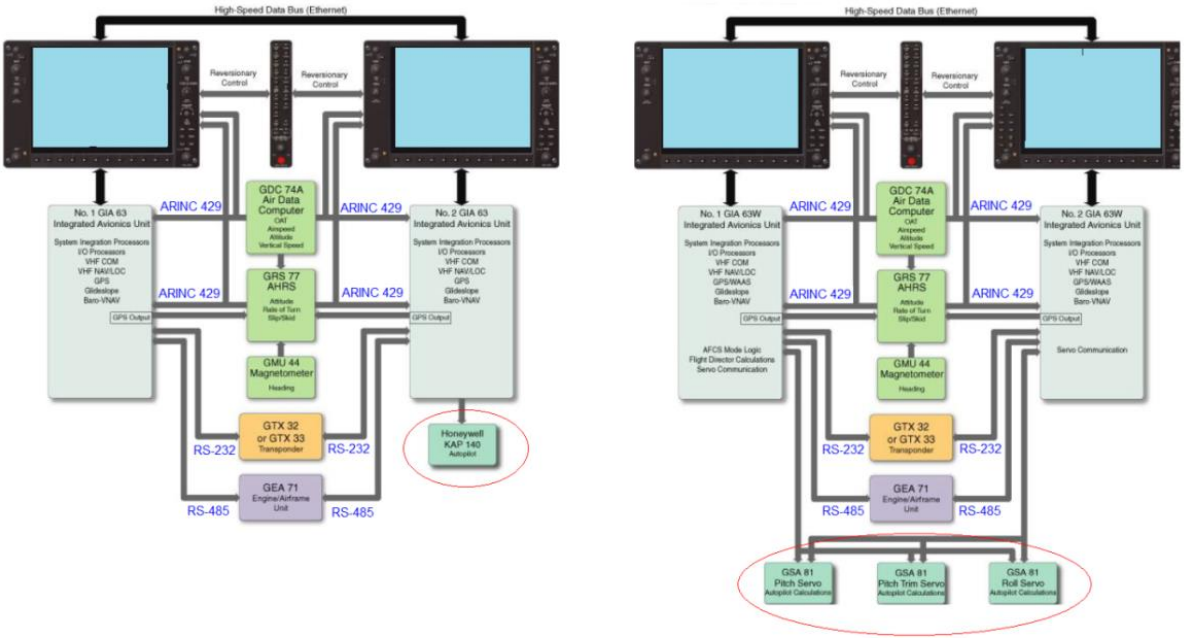


Figure 2:4 EFIS Autopilot Wiring Diagram [20, p.22].

Recent Commercial avionics systems became more modular and flexible. They integrated numerous units using bus interfaces where the autopilot is only one more device inside the architecture. For instance, the Garmin Company developed an integrated avionics system, the G1000 [21] which incorporated the communication, navigation, surveillance systems, air-data unit, engine interfaces, and heading Reference System in line-replace units. G1000 comprises two liquid crystal displays; (GDU 1040) one is the Primary Flight Display (PFD) and the second is the Multi-Function Display, two GIA 63 Integrated Avionics Units (IAUs), the GRS 77 AHRS and one audio panel. The GIA 63 integrates additional components into G1000 like the GDC 74A which receives the signals from Pitot, Static and air temperature inputs. Two GDU 1040 displays and two GIA 63 IAUs communicate between them using a proprietary Ethernet-based protocol. G1000 also supports different serial interfaces such as RS232, ARINC 429 and RS-485 (Figure 2.5).



(a) (b)

Using autopilot KAP 140 [21, p.2-17] Using autopilot GFC700 [21, p.2-18]

Figure 2:5 Block Diagram Autopilot Garmin G1000

There are two autopilot models compatible with G1000 one is G700 developed by Garmin and KP 140 [22, 23] developed by Bendix/King (Figure 2.5a). Both autopilots G700 and KP 140 must be integrated to G1000 using servos produced by its same manufacturer for compatibility. The reason is that the group of servos GSA 81 work with G700 sharing a digital proprietary bus interface. The architecture of the model KAP 140 interfaces its group of servos with analogue bus, similar to model EFIS AP101 [20]. Figure 2.6 illustrates how each servo is connected to autopilot KAP 140 through line controls according to its type. The second reason is that the position and speed control for servos run inside the autopilot, which means both KAP 140 and G700 have been tuned to work with its own servos. However, Garmin does not provide information about if there is a position control running into each servo or into the G700 autopilot.

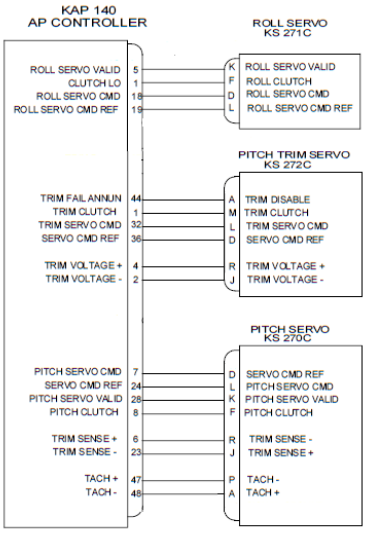


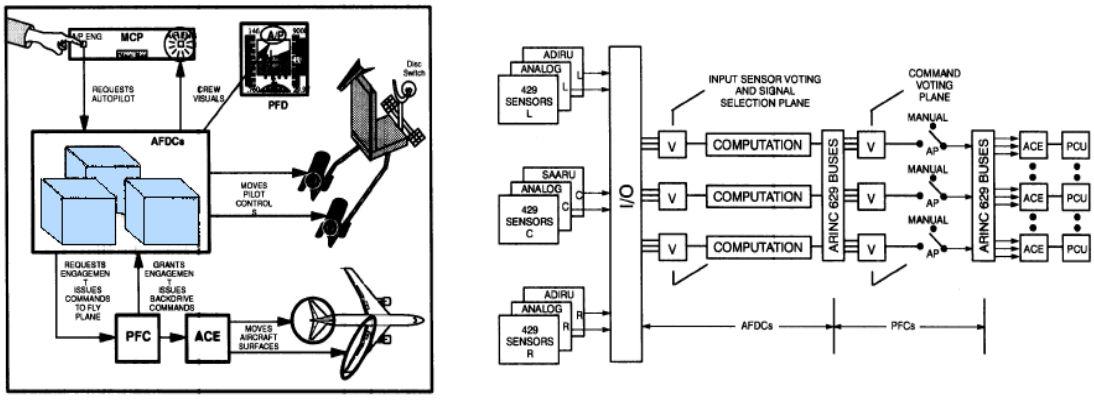
Figure 2:6 Diagram of KP 140 wiring with servos. Adapted from [120].

An analysis of autopilot models MGL Avionics, GRT, KP 140, EFIS and Garmin GFC 700, confirms that servos is avionics part difficult to standardise because each manufacturer matches the autopilot with the servos. However, MGL Avionics solves this problem of incompatibility with their MGL Autopilot Servo [24], as shown in Figure 2.7, with two kinds of bus interfaces using the same DB9 connector: one being the CAN interface and the other, RS232 which it allows interfacing autopilots with one of these interfaces. MGL Autopilot Servo can be configured to use one of these interfaces but not both at the same time. The important fact is not only that this servo has two interfaces, but also the servo manufacturer provides the technical details about the protocol implementation [25]. A similar servo is produced by the French avionics company SAGEM known as SM1000 which has an improved version of the CAN protocol which specializes in avionics data [26].

Figure 2:7 MGL Servo [24, p.12].

For commercial aircraft like Boeing 777 the avionics hardware is often customized for each aircraft. According to Hornish [27], the Boeing 777 uses an Autopilot Flight Director System (AFDS) which has three modules: one is the Mode Control Panel (MCP) where the crew can select the

autopilot vertical and lateral modes, three Autopilot Flight Director Computers (AFDCs) which support the operations for the flight director, autopilot, and maintenance, and backdrive functions; and six backdrive control actuators (BCA), as illustrated in Figure 2.8. These BCA send feedback commands to crew indicating the column and pedal position when controlled by the autopilot. The three AFDS work in a redundant system to protect the aircraft again a failure where each AFDC has its own group of sensors.



(a) AFDS Control of the Aircraft (b) Frontdrive System Architecture

Figure 2:8 Autopilot Flight System for Boeing 777 [27, p.152]

All sensors and MPC use a bus system ARINC 429 to low speed (12.5Kbps) to communicate with AFDCS. The AFDC controls the aircraft flight surfaces also sending commands via ARINC 629 to Actuator Control Electronics Unit (ACEU) which interface the servos with the system and the electrohydraulic power control units. The hardware used for every Autopilot Flight Computer is a processor FCP-2000 which was exclusively designed to work for this flight control system executing Ada code.

2.3 UAS AUTOPILOT ARCHITECTURE

UAS applications are becoming more complex and require more processing capacity, sensors and actuators to execute their applications. The UAS autopilot hardware architecture needs independent units of processing to execute the navigation and guidance algorithms, image processing, and to interface with multiple sensors and actuators. The microelectronics devices are also becoming smaller and smarter and power consumption lower which allow being installed on board to UAS.

These requirements cause UAS autopilot designers to develop modular designs with different architectures, some UAS base their hardware design on one high-performance microcontroller; others

use two or more microcontrollers in a master-slave configuration. Some UAS autopilots use one or a combination of DSC, DSP, and FPGA when their applications are based on algorithms which demand high processing or the UAS AFS designers want to separate the tasks in different specialized units. Some architecture use CAN interfaces to connect different devices on-board UAS, whereas others projects implement technologies related to distributed systems.

2.3.1 UAS Architecture based in Microcontrollers

The standard UAS AFS design used to develop the hardware architecture is based on a unique microcontroller using the low-level bus interfaces SPI, I^2C and RS232 to connect the on-board sensors (Figure 2.9). An example of this type of design was proposed by Jung et al. in [28] who implemented a UAS AFS using a Rabbit microcontroller and connected the different sensor devices with SPA and serial communication, as illustrated in Figure 2.9. The set of servos receives PWM signals using I/O pins from the microcontroller. The prototype was developed for the Georgia Institute of Technology for academic purposes. The system was designed for one type of airframe and the system is highly-coupled, which means it does not have an ideal level of modularity.

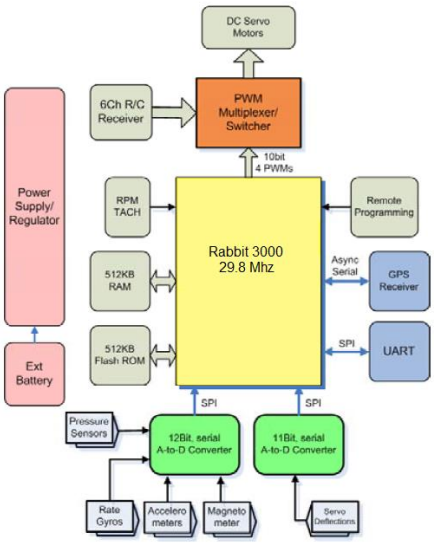


Figure 2:9 Standard Hardware Design for UAS Autopilot [28 p.2740].

An autopilot hardware architecture using a configuration master-slave comprised by two 8-bit microcontrollers (RCM410) was designed and implemented by Lara et al. [29]. The master unit executed the autopilot core tasks and the slave unit supported the reading of three ultrasonic sensors to know the distance from the helicopter to a target (Figure 2.10). The tasks for both microcontrollers are executed in parallel and the communication between both units is serial. This design is highly coupled and its modularity is low as the autopilot was implemented only for a quadcopter model. A similar approach was proposed by Maranhao and Alsina [30] who presented a master-slave design for a micro UAS using an embedded computer like a master which runs the control and image processing and two

microcontrollers, one to interface the sensors, and the other to interface the servos. The communication between the computer master unit and the microcontrollers is via a USB port.

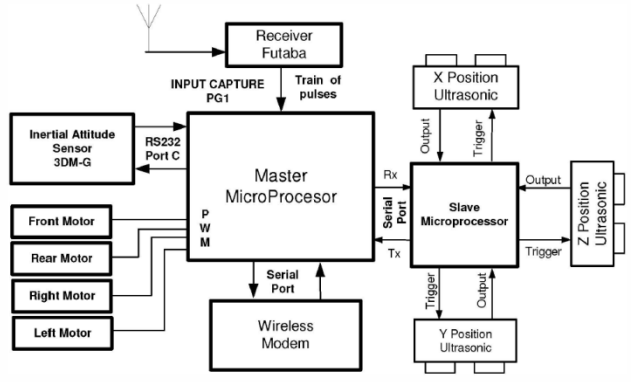


Figure 2:10 Autopilot Hardware Architecture using a configuration master-slave [29, p.2].

Rogers et al. [31] developed an adaptive autopilot system for small Fixed-Wing UAS called Reconfigurable Autopilot for Vehicles with Enhanced Navigation RAVEN. This system was designed to use with different airframes and combines an adaptive control algorithm and hardware interfaces to recognize the UAS models. The hardware architecture is composed of two Flight Processing Units (FPUs) which separate AFS functions from sensor data acquisition.

Atkins, Eubank & Klesh [32] created an adapted an open-source platform to implement a reconfigurable flight management system for three types of Fixed-Wing UAS. This project has a complex hardware structure where the flight control algorithms (guidance, navigation and control) run in a master Gumstix board and the data acquisition runs over a Atmel-based Robotix daughter board. Both boards communicate using an I2C interface where the master unit runs an embedded Linux kernel and the Robotix software was developed in C with AVR cross compiler (Figure 2.11).

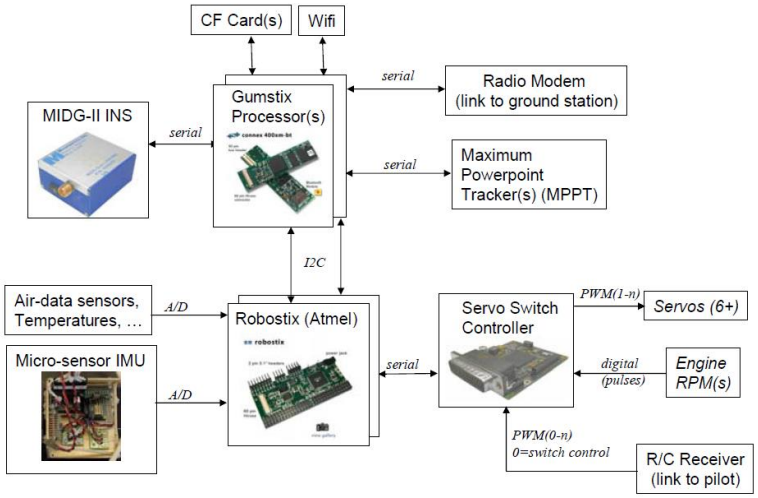


Figure 2:11 Flight Management System Hardware [32, p.3].

Tretyakov and Surmann [33] proposed a modular autopilot for a quadrotor which can communicate with a ground station via analogue radio link or through a Bluetooth connection with a smart phone, table or laptop. The design was based on open-source autopilot MikroKopter which was modified to integrate a Bluetooth Free2move board and a Gumstix board to communicate with ground station via analogue radio link. This project was focused to separate the communications module from the guidance and control module.

Ax et al. modified a commercial Mini Unmanned Aerial Vehicles (MUAV) system called MD4-200 used to work with quadrocopter [34]. MD4-200 was modified to improve the level of autonomy increasing the responses to internal and external events without requiring mediation from a ground station. The hardware modification included the addition of a microcontroller ARM9 with a speed processing of 200 MHz, SDRAM of 64MB and 2GB microSD card and the software installed included a GNU/Linux distribution. The microcontroller is used like Bridge which receives and transmits data from ground station using IP protocols and sends and receives data from MD4-200 using serial communication (Figure 2.12). The prototype was tested connecting a series of sensors such as optical flow sensor and CMOS image sensor using a CAN Bus provided by the additional microcontroller.

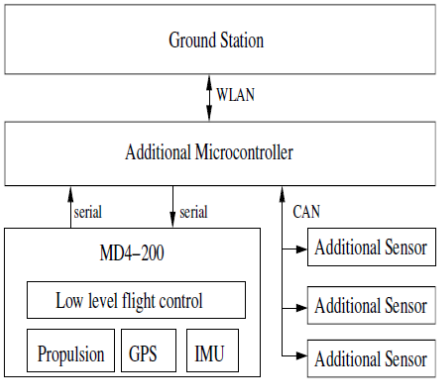


Figure 2:12 Hardware Architecture of modified MD4-200 [34, p.849].

2.3.2 UAS Architecture based on DSC

A reconfigurable UAS autopilot developed by Lizarraga et al. [35] created an autopilot design supported by two Digital Signal Controllers (dsPIC33F). One DSC runs the control core algorithms, reports the telemetry to ground station and generates PWM signals to servos. The other DSC executes the reading of sensors and performs attitude estimation. Both DSCs are communicated using the high-speed Serial Peripheral Interface (SPI) bus, as illustrated in Figure 2.13. The pressure sensors are read using ADC; the GPS transmit its data via serial communication. Also, this design allow adding new sensors through CAN protocol.

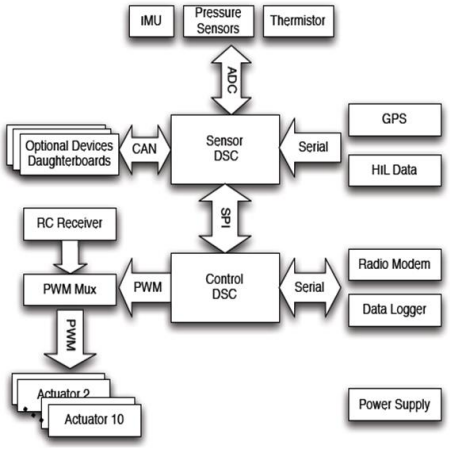


Figure 2:13 Autopilot Block Diagram Using two DSC [35, p.4].

A similar approach but only using a DSC on the UAS autopilot design was presented by Haifeng and Xiajing to implement a small UAS autopilot in [36]. This design uses the DSC (TMS320F2812) to run the control and navigation algorithms and to interface the sensors and actuators. Also, the DSC establishes the communication with the ground station.

2.3.3 UAS Architecture based on DSP and FPGA

An UAS autopilot using a DSP and high-performance microcontroller was implemented by Cunxiao and Jiancheng in [37] for a low-altitude mapping system on a Small Unmanned Aerial Vehicle (SUAV). This design used a DSP (ADSP21364) to run the complex algorithms of navigation and control and it used a microcontroller (AT91RM9200) to obtain the data from sensors, command servos and communicate with a ground station. Cunxiao and Jiancheng developed this high-performance hardware autopilot to get high level navigation precision using two GPS units (Figure 2.14). The GSP sent data via USART, the ADCS communicated to microcontroller via SPI bus interface, the protocol used to communicate the DPS and microcontroller was not mentioned. DSP-based AFS architecture was developed by Luo and Chen [118] who designed a data acquisition and reliable transmission of micro-UAV using a DSP (TMS320F2812). The DSP executes the all autopilot tasks and interface all sensors using the RS232 and PSI interfaces.

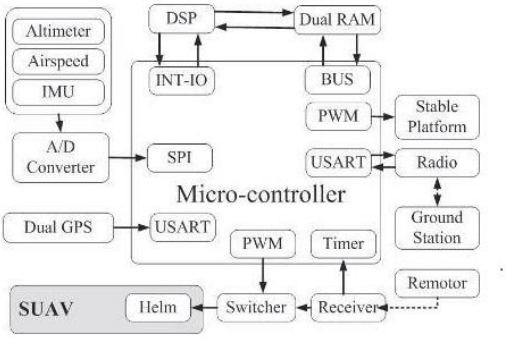


Figure 2:14 UAS Hardware Architecture using one DSP and one microcontroller [37, p.1745].

Christopher et al. proposed a cross-platform [38] to design a UAS autopilot for small UAVs using an integrated Flight Control System (FCS20) which includes a FPGA (Altera Stratix II) to interface the external units (sensors and actuators) and a DSP (TMS320C6713) to run the guidance, control, navigation algorithms (Figure 2.15). A MicroC/CO-II RTOS runs over the DSP and manages all the function and tasks of an Operating System and supports the flight control libraries. The communication between the DSP and the FPGA is done through a 32-bit parallel bus to 250 Mbps. Because of FPGAs can be reprogrammable to build a specific hardware inner design [40] these can be re-configured to work with new sensor. GPS uses a serial to communicate with FPGA and gyros, attitude, airspeed and temperature sensors use the SPI interface to transmit the data.

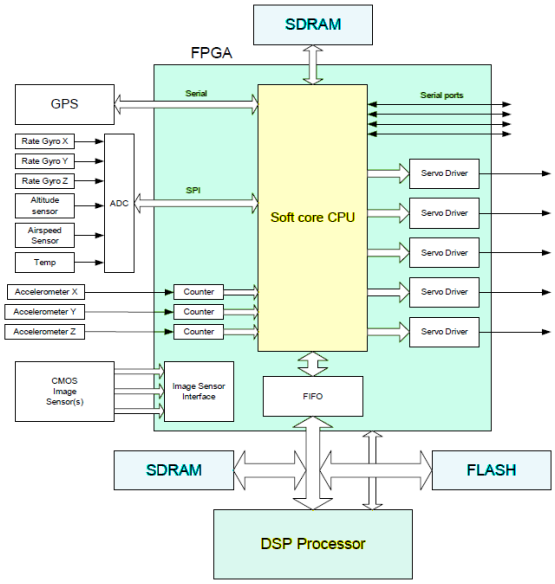


Figure 2:15 Hardware Design using a FPGA for I/O connection and a DSP for signal processing algorithms [38, p.3].

Alvis et al. [39] developed a FPGA-based autopilot design to integrate with different UAS

platforms taking advantage of parallel processing and the modularity to communicate analogue sensors that a FPGA supports. The system is comprised of a slave board (Xilinx Virtex II Pro FPGA) used to interface all sensors and a DSP-based master board (PC-104) where the vision, mission planner and communication modules are executed. The communication protocol used between the two boards is USB and to communicate sensors they used the serial interfaces RS232 and USB.

A similar hardware platform was used by Scherz et al. in [40] who developed pulse radar target detection for a UAS. The FPGA (Cyclone III) executes all autopilot functions including the reading of radar data whereas the DSP (TMS320C6713) computing the radar processing algorithms using the radar data obtained by the FPGA. The data transfer between the FPGA and DSP is established using a 16-bit external memory interface (EMIF). Also, the FPGA has a functionality to map peripherals which manages the SPI and UART interfaces to communicate the group of sensors.

A helicopter platform was presented by Ragavan et al. in [41] who designed a based-FPGA board using a Cyclone III with 32 MB of SDRAM, 16MB of flash memory. This FPGA is used to run both the navigation and control algorithms and to read all data sensors and transmit the PWM signals to actuators. At the same time, Klenke in [42] proposed anUAS hardware architecture comprises for a FPGA which generates the PWM signals to servos and establishes the communication with the ground station and an AVR microcontroller which executes the control and navigation algorithms.

2.3.4 UAS CAN-based Architecture

Spinka et al. [43] created an open-source autopilot system for rotorcraft UAVs using a networked hierarchical distributed control system called Remotely Operated Aerial Model Autopilot (RAMA). This system was comprised by three main modules: Main Control Computer (MCC), the Navigation Unit (NU) and the Servo Control Unit (SCU) interconnected by a CAN Network to 100Kbps (Figure 2.16). MCC is the autopilot core module where the control and communication programs run over a microcontroller Renesas SH7760 using GNU/Linux. The NU runs over a microcontroller Philips LPC2119 and receives the data from three units GPS, IMU and Three Axis Magnetometer TAM via RS232. Finally, the SCU runs over a board Renesas 2638F MCU and controls the servo actuators without using an operating system.

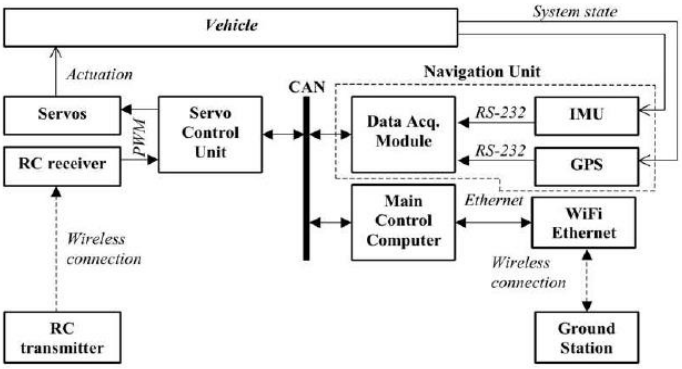


Figure 2:16 Remotely Operated Aerial Model Autopilot (RAMA) block diagram [43, p.882].

A UAS autopilot using the protocol CANAerospace, was proposed by Catena et al. in [44] who divided the system in three sub-units sharing the CAN Bus. The main subunits are Air Data Attitude Heading System (ADAHRS) which read the data from the GPS, the Inertial Measurement Unit (IMU) and the air data sensor (Pitot - static tube). Other subunit is The Flight Control Computer System (FCCS) where run the navigation and stabilization algorithms and the SACS Servo Actuator Control System which manages the group of servos. The hardware used to run the software system is: an ARM microprocessor to execute the ADAHRS unit, a 16-bit DSP to run the FCCS unit.

2.3.5 UAS Architecture based on Distributed Systems

Few UAS AFS implemented a distributed architecture using sensors, actuators and speed modules which send and receive data from to an autopilot using interfaces. To implement the distributed architecture was necessary to develop a middleware technology. Middleware is based on a layer of software used to interface the functions executed by the operative system, the UAS applications and the communication with sensors and actuators. The middleware layer is implemented for each element into Network to provide services which can be used by other elements in the same Network. One of the technologies proposes to develop a distributed architecture is Service- Oriented to Architecture (SOA).

D. Zhicheng et al. in [45] designed a Reconfigurable Flight Control System Architecture for Small Unmanned Aerial Vehicles (RFCSA) which was implemented in SOA to develop an AFS. One of the aims of this project was to reuse hardware and software components to port to another application without adjustments where each avionic component requires a software and hardware interface to bring a set of generic services (Figure 2.17). These services can be called by any application running inside the autopilot using a SOA implementation without knowing technical aspects about the interfaces.

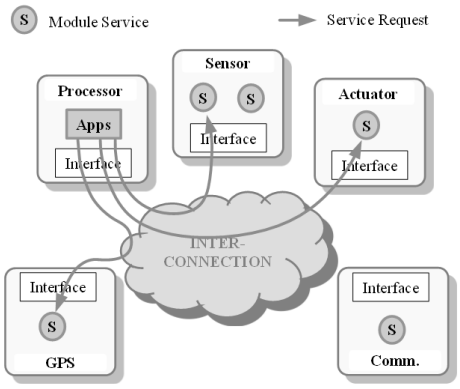


Figure 2:17 SOA enabled RFCSA [45].

A related work using SOA was created by Santamaria et al. [46] who introduced a Flight Control System Gateway (FCSG) which bridges the autopilot with the other systems in the UAS using a LAN network, as shown in Figure 2.18. This avoids an aerodynamic redesign in the UAV that every time that a new autopilot is installed. The autopilot receives and sends data from/to FCS Gateway and at the same time the FCS Gateway transmits and receives the data from to others flight systems in the UAV. FCSG needs to be configured to each type of autopilot but it allows that all systems can be reused without new modifications.

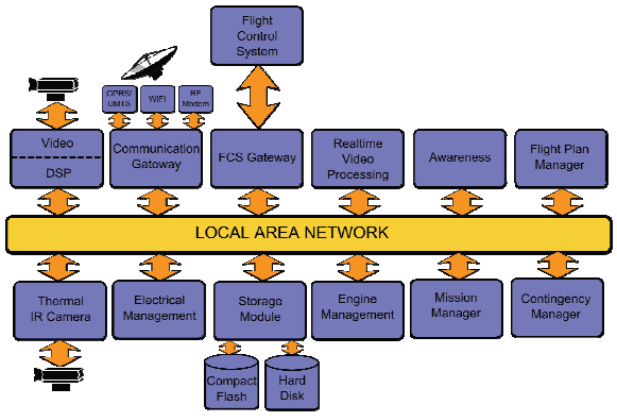


Figure 2:18 Overview of the USAL Service-Based Architecture [46, p. 5.B.5-3].

Using the concept of distributed systems an alternative technology to SOA is CORBA (Common Object Broker Architecture) used to communicate different types of systems running on multiples platforms. This technology was employed by Doherty et al. in [47] to be used in prototype UAS system based on RMAX Helicopter. The hardware and software design is comprised of three embedded computers PIII (700MHz) with high-processing capacity running GNU/Linux and interconnected to an Ethernet Network. One computer is in charge to connect the some UAS sensors via serial bus or using ADCs. The second computer runs the image processing algorithms and the third executes the control and guidance algorithms. Similar to SOA, every embedded computer shares services which are called by the others computers to execute their tasks.

2.3.6 Commercial off-the-shelf and open-source UAS autopilots

Multiple commercial and open-source autopilots have been developed with the aim to be reused for multiples types of rotary-wing or fixed-wing UAS without requiring major adjustments. The COTS autopilots do not bring details about their software and hardware designs and only technical information is deduced of its user catalogues. Whereas open-source autopilots provides all information about their hardware and software designs which allows modifications in their designs to develop applications. Both use the same kind of serial interfaces such as RS232, SPI, I^2C and recently CAN to connect the group of sensors such as GPS, gyros, accelerometer, pressure, infrared and vision, etc. Also, the majority of UAS servo manufacturers developed their servos to work with PWM signals using a frequency of 50 Hz which allow a simple integration with any autopilot hardware.

The predominance hardware design for open-source and commercial UAS autopilot is based on high-performance microcontrollers where all autopilot functions, the servo interfaces and the reading of sensors are executed (Figure 2.8). For instance, the series Piccolo II [48] uses a 32-bit Motorola processor; Procerus Kestrel uses a 29 MHz Rabbit 3000 [49]; Paparazzi autopilot [50] employs nine different hardware designs using microcontrollers such as ARM7 and SMT32. Similar to Paparazzi, Pixhawk [51] works with a 32-bit microcontroller STM32F427 Cortex M4 core with FPU. An exception to use high-speed microcontroller is the Ardupilot [52] which uses a low processing 8-bit microcontroller ATMEGA328P. Other hardware autopilot platforms which use an only microcontroller in their hardware design are Arducopter based on Atmega2560 [52], Openpilot [53] (STM32F103CB). There are two commercial autopilots which employ a more complex hardware structure. One is the Microbot autopilot [54] is designed over FPGA to implement the logic for external components and one microprocessor where the guidance and control programs are executed. The second commercial design is the PC-104 [55] board which includes a DSP and microcontroller.

2.4 SUMMARY

Many projects for both general aviation and UAS have been developed thinking in modular designs for avionics equipment. Protocols such as ARINC 429, ARINC 629 and CAN-based protocols and have been implemented to ensure the interoperability between different avionics systems including AFSs. This has facilitated that AFSs can be configured for different aircraft. Due to CAN interface has high levels of reliability to detect errors and brings high electrical and magnetic immunity and flexibility to integrate new components has become to develop modular hardware and software applications. Some avionics manufacturers work integrated modular units such as G1000 (Garmin) which can adapt new elements and can include autopilots developed by other manufacturers such as KAP 140 (Bendix). However, often the group of servos cannot be appropriately integrated to work with other GA autopilots. The majority of the GA autopilot models analysed during the literature

review would not allow being installed on-board of an UAS because the servo connections use property interfaces and do not provide PWM signal which are required by UAS servos. However, the MGL Avionics autopilot models provide a PWM port to connect servos (Figure 2.1), but the power supply would avoid connecting UAS servos because GA Aircraft servos work to 28 Volts whereas the UAS servos work to 5 Volts.

UAS have created different hardware architectures to achieve high interoperability. These prototypes have divided the autopilot functions to execute in specialized microprocessors such as microcontrollers, Digital Signal Controllers, Digital Signal Processors and FPGA. Generally, the autopilot modes and reading of sensors are executed in a microcontroller and digital signal processing algorithms are executed in DSP. Others projects propose to execute the guidance and navigation algorithms in a DSP and the reading of sensors using SPI, I^2C or RS232 interfaces and PWM generation in FPA due to its capacity parallel processing. Other researches combine different types of microprocessor in master-slave mode. Only two projects modify existing autopilot designs to increase their performances or customize an UAS application. Unlike to servo interfacing in GA aircraft, manufacturers for UAS servos built their servos with similar features which allow compatibility with any type of autopilot. Similar situation happens with the sensors which can be interfaced with different autopilots using serial interfaces (SPI and I^2C) or using ADCs.

Two UAS autopilot projects propose the use of CAN protocol to interface different components on-board of UAS and three projects implement strategies to adapt distributed systems into UAS autopilot systems. The use of CAN protocol increases the possibilities to make more modular an UAS AFS and take advantages that this protocol offers in environments with high level of mechanical and electrical noise such as UAS. The alternative to use distributed architectures using middleware software in UAS is interesting but it is not convenience. This type of Architecture requires layers of software and high-performance processors which increases the use of resources for AFS processing.

The literature review does not present an AFS which can be used for both GA Aircraft and UAS. The key point is the use of CAN interface which is used in general and commercial aviation and until now only has been used to integrate servos or sensors on-board of UAS. The GA and UAS autopilots are specialized to design prototypes for manned or unmanned aircraft but not for both. This research proposes a new architecture which integrates a UAS autopilot system with servos used to work in a Cessna Aircraft and allows UAS autopilot to be used for both GA aircraft and for UAS. This opens the possibilities in a future that UAS systems can be integrated into GA aircraft reducing costs because UAS autopilot are less expensive than GA autopilots. Also, this allows the development of new avionics applications which can be used for manned and unmanned aircraft.

Chapter 3: Reconfigurable Autopilot Flight System Design

This chapter presents the 3D Reconfigurable Autopilot Flight System to be used for both a GA Aircraft and UAS. The development of an entire autopilot would imply a long development process, time and investigation this, research used an existing UAS autopilot system which was robust, open source and widely used in research projects. In this way, this research designed, developed and implemented an interface to integrate this UAS autopilot with a GA aircraft servos and UAS servos.

The design and implementation of this AFS has been divided into three independent and modular parts, as illustrated in Figure 3.1. The first part is the UAV autopilot flight system Pixhawk (Px4) which will be known as the Front-End module and will transmit position commands to servos using CAN Network. The second part is the Bridge module which receives the position commands from Px4 and generates PWM signals to servos. The third part is the Back-End module used to interface the servos for both UAS and GA aircraft. Each of these three modules will be described in terms of its hardware and software components.

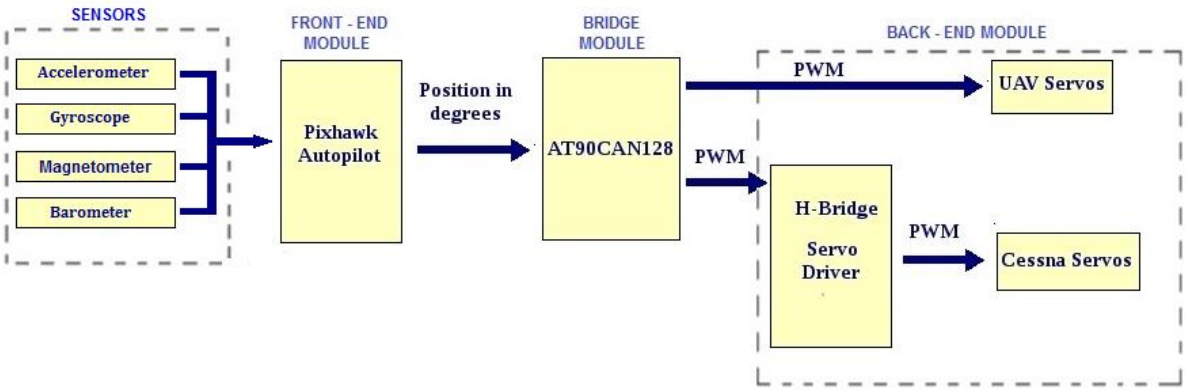


Figure 3:1 Block Diagram of Reconfigurable Autopilot System (RAFS)

3.1 RAFS HARDWARE ARCHITECTURE

The design of RAFS hardware architecture is comprised by the open hardware design and open source Pixhawk autopilot which is the main component of Front-End module. The board OLIMEX AVR-CAN is the core of the Bridge module and used to interface the autopilot flight system with the Fixed-Wing UAS and Servos used on a GA Aircraft. The Back-End module is a combination of three UAV servos: Pitch Servo HS-645MG, Roll Servo HS-985MG and Rudder Servo HS-645MG, two H-Bridge modules MC33926 and two Cessna Servos Pitch Servo SE-816A and Roll Servo SA-816D (Figure 3.2). Two CAN transceivers are necessary to connect the Front-End and Back-End modules through CAN bus; both are included in the Pixhawk autopilot and AVR-CAN board.

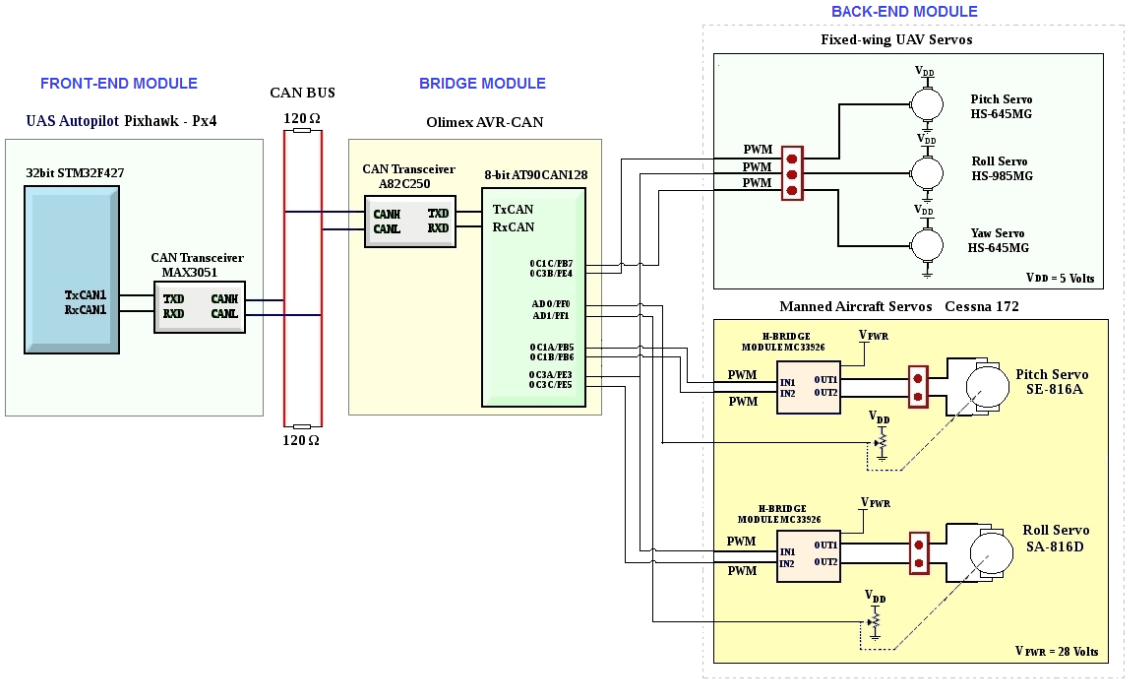


Figure 3:2 Hardware Architecture for RAFS

The hardware architecture shows two main physical connections. The first one is the connection between the Pixhawk autopilot (Front-End module) and the Olimex AVR-CAN Board (Bridge module) via CAN Bus version 2.0A to 250 Kbps. The second connection is between AVR-CAN Board and the set of servos (Back-End module) using the I/O port pins to send PWM signals and two pins to read the Cessna Pitch and Roll Servo positions using two Analogue Digital Converters (ADC).

3.1.1 Front-End Module and Interface

The front module is based on the PX4 Pixhawk Autopilot developed by the Computer Vision and Geometry Lab of ETH Zurich (Swiss Federal Institute of Technology), Autonomous System Lab and the Automatic Control Laboratory. The PX4 Pixhawk project aim is to develop an independent, open-source, open-hardware autopilot platform which can be used in different UAS applications such as multicopters, small fixed-wing UAS and rovers. The main advantage to PX4 Pixhawk is an open hardware and open-source platform is that it allows studying, adapting or modifying the autopilot to create new applications or add new functionalities. This section describes the hardware and software features of UAS Pixhawk autopilot.

3.1.1.1 PX4 Pixhawk Autopilot Hardware

The Pixhawk autopilot core is a 32-bit ARM Cortex microcontroller STM32F427 [56] with FPU which is optimal to run some complex embedded applications as it has an high speed processing up to 180 MHz, a flash memory equal to 256KB enough to store medium size programs, and RAM memory equal to 2MB. Also, STM32F427 may work using low-power-consumption modes (Sleep, Stop and Standby) and has an interesting set of DSP instructions to work image processing, multiples I/Os and peripherals. Other features that STM32F427 includes are three 12-bit ADCs, two DACS, twelve general-purpose 16-bit timers including two PWM timers, and two 32-bit timers. A detailed Pixhawk schematic and layout is presented in Appendix E.

A remarkable characteristic is that STM32F427 has a high number and type of interfaces such as I2C, SPI, I2S, five USARTs, Ethernet and two CANs ports which ensure interconnectivity with different sensors or actuators. All these interfaces have been implemented on PX4 Pixhawk Autopilot (Figure 3.3). However, only one CAN interface has been adapted to Pixhawk Autopilot using a 3.3V transceiver whereas the other CAN interface would require implementing supplementary hardware.

Pixhawk [51] is a low cost autopilot compared with manned aircraft autopilots and it has physical features like small dimensions (50mm X 15.5mm X 81.5mm) and light weight (38 gr) which allows it to be installed on GA aircraft or UAS. The Pixhawk operating voltage is around 4.1 to 5.7 V, where the ADC inputs can work in range of voltages between 3.3 and 6.6V.

Figure 3:3 Pixhawk Autopilot Interfaces [51].

As it PX4 has two CAN ports, the Pixhawk was selected as one of essential parts to develop this project. Using CAN interfaces, an Unmanned Autopilot System can interoperate with a bridge module to interface Human Machines Interfaces (HMI), sensors or servos used in a GA aircraft.

3.1.1.2 PX4 Pixhawk and CAN Transceiver

The CAN transceiver (transmit/receive) is an essential device to interface physically a microcontroller with CAN bus lines CAN_H and CAN_L, detailed in Appendix B. This device converts the binary representation into differential signals of voltage when transmits and receives information to/from a CAN bus. When PX4 transmits a CAN message, the transceiver must provide enough current to change the bus electrical state, must avoid signals reflections and must reduce the electromagnetic interference. At the same time, the CAN transceiver avoids high voltages or short circuits that can destroy the Pixhawk due to electrical isolation. Px4 includes the CAN transceiver MAX3051 [57] used to connect the Px4 CAN port 1 with a CAN network (Figure 3.4).

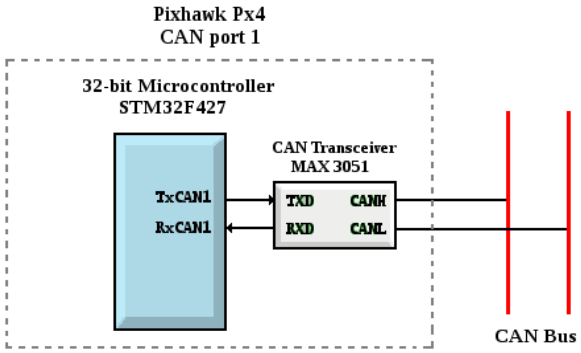


Figure 3:4 Diagram of SMT32F427 and CAN transceiver used in Pixhawk Autopilot.

3.1.1.3 PX4 Pixhawk Autopilot Software

PX4 Pixhawk autopilot has been developed and implemented using open source software tools. The philosophy behind an open source project is that the effort of many developers can increase the functionalities and find bugs in the code source. Pixhawk module runs using Nuttx RTOS and uses a development environment based on cross toolchain. The following section describes some features about Nuttx RTOS and the GNU toolchain used to implement UAS applications on Pixhawk.

3.1.1.3.1 Nuttx Real Time Operative System (RTOS)

Pixhawk runs under the open-source real-time operative system known as Nuttx developed in C, which is highly configurable and extendible to new processors and boards. The first Nuttx version was released in 2007 and follows the POSIX [58] and ANSI standards which means features such as concurrency, clocks, timers, command interpreter, shared memory, multithreading, exceptions and drivers have been developed ensuring compatibility between Unix-like operating systems. In like-Unix System, Nuttx runs Pixhawk applications using shell commands (NuttShell) which can be executed remotely from a computer terminal using the serial protocol Kermit via USB.

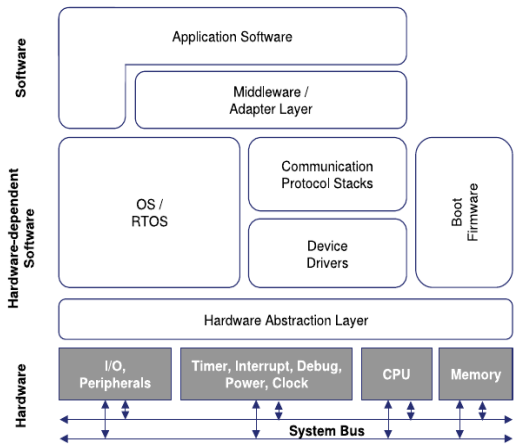


Figure 3:5 Layered Software Architecture [59, p.9].

Due to its modular design based on layered software architecture [59], as shown in Figure 3.5, Nuttx can work with different 8-bit and 32-bit microcontroller families such as ARM, AVR, Intel, Renesas and Zilog. This modularity allows Nuttx to include a series of drivers used to operate with diverse types of devices such as analogue, graphic and input devices network such as USB, RS232, I2C, I2S, NAND, CAN, ADC, DAC and PWM. Nuttx has a library collection to work different protocol stacks including TCP/IP, UDP, ICMP and raw sockets.

Pixhawk applications are placed on Nuttx user space where autopilot functionalities like UAS

autopilot modes are executed whereas Nuttx kernel activities like interrupt handlers, concurrency and memory management run on kernel space. Thus if one Pixhawk application fails it does not affect RTOS Nuttx tasks. All these Nuttx features allow Pixhawk developers to focus their effort on UAS applications without worrying about low level details related to software and hardware.

3.1.1.3.2 Pixhawk Cross Toolchain

Pixhawk Autopilot supports its development software using a set of open-source tools to compile, link and generate the executable code that will run into UAS autopilot hardware. Additionally, GNU/toolchain includes debugger or emulator tools to analyse the program logic and detect errors and software tools to upload the firmware into a flash memory. To develop Pixhawk embedded applications require using a cross compiler because the build machine where the code source is being developed has different hardware architecture respect to target machine where the embedded application will run [60]. For instance, if the Pixhawk code is developed on x86 running GNU/Linux, it will be necessary to install a cross compiler to generate the code for the microcontroller STM32F427. The toolchain used to create Pixhawk applications in C/C++ is the GNU programming platform for ARM Embedded Processors, specifically Cortex-R/Cortex-M processor families [61].

3.1.1.4 *Interface between Front-End Module and Bridge Module*

The physical connection between the Front-End and Bridge modules is implemented using a shielded twisted-pair CAN cable with a two 120Ω resistors at both ends. According to Standard ISO 11898, these resistors avoid that signal reflections could be transmitted at any node connected to CAN Network [62]. The CAN cable connects two transceivers, one side of the Pixhawk Autopilot and the other side of Bridge module through CAN high voltage wire and CAN low voltage wire, as described in Appendix B.

3.1.1.4.1 CAN Network Topology

The network structure was designed using two physical CAN nodes, the main node is the Px4 UAS autopilot (Front-End Module) which has the highest priority on the CAN network and the second CAN node is the Bridge Module. The CAN node used on the Bridge interface is configured to work with five CAN logic nodes (Figure 3.6). Two logic nodes correspond to Cessna Servos, the Pitch Servo SE-816A and the Roll Servo SA-816D; the other three nodes correspond to Wing-fixed UAS servos, Pitch Servo Hs-645MG, Roll Servo HS-985MG and the Yaw Servo HS-645MG. All CAN frames sent from the Px4 autopilot are driven by the Bridge Module and redirected to each one of the

servos. Chapter Four details this logic and additional information about CAN nodes is detailed in Appendix B.

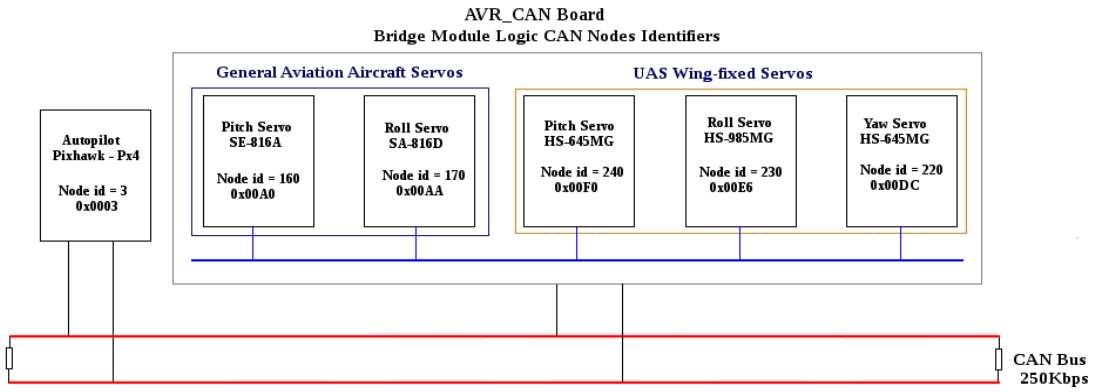


Figure 3:6 CAN Bus Network RAFS using two physical nodes and five logic nodes inside Bridge module.

3.1.2 Bridge Module

The Bridge module is a hardware and software interface comprised of board Olimex AVR-CAN which receives the position commands from Pixhawk via CAN Network and transmits PWM signals to UAS or Cessna servos depending the system configuration. If the system has been configured to work with a Fixed-Wing UAV, the interface sends PWM signals to UAV servos, but if the system has been configured to work with a general aviation aircraft the system transmits PWM signals to Cessna Servos. The bridge increases the Pixhawk modularity, isolating the autopilot functionalities of control logic to move servos. This modular design allows an UAS autopilot to integrate elements used on a manned aircraft which could make it possible that Pixhawk can be adapted on board a GA Aircraft.

3.1.2.1 *AVR-CAN Hardware*

The Olimex AVR-CAN [63] is a low-cost board optimal to interface hardware applications using CAN protocol or RS232 interface (Figure 3.7). AVR-CAN consists of 8-bit AVR microcontroller AT90CAN128 [64] designed under RISC architecture with low-power consumption. This board has a 128Kb programmable flash memory, 4Kb EEPROM, 4Kb SRAM, six 8-bit I/O ports and one I/O 5-bit port, 32 general purpose working registers, four Timers/Counters with PWM, eight 10-bit ADC. Also, Olimex AVR-CAN offers connectivity with SPI serial port, JTAG interface, RS232 and CAN protocol through transceiver A82C250. CPU speed works at 16 MHz, using operating voltages between 7 and 12V and current consumption about 40-50mA. A detailed AVR-CAN board schematic is provided in Appendix E.

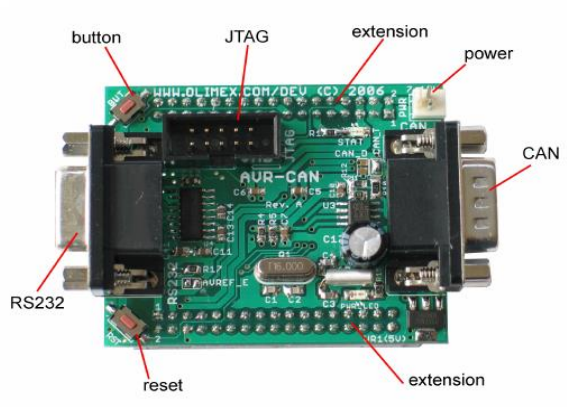


*Figure 3:7*AVR-CAN Board [63].

3.1.2.2 *Olimex AVR-CAN Software*

One aim of this research project is to use open hardware and open source software as extensively as possible. The board Olimex AVR-CAN is designed to work with the Integrated Development Platform (IDP) AVR Studio 4.13 developed by ATMEL and the open source software development tools for AVR microprocessors WINAVR based on GNU GCC compiler. Although AVR Studio can be downloaded freely, this is not open source software and its license has copyrights [65]. Additionally, AVR Studio can only be installed in Windows [66] meaning that cannot be used in General Public License (GPL) [67] Operating Systems such as GNU/Linux distributions or FreeBSD.

Alternatives to develop the Bridge module software for AVR-CAN board using an open-source Integrated Development Environment (IDE) and Atmel libraries with GPL license over Linux were evaluated. The viable alternative was to develop source code using the java-based Arduino platform IDE [68, 69] which can be installed in multiple operating systems. Similar to Pixhawk, Arduino IDE requires the GNU AVR cross toolchain to compile (avr-gcc), simulate (avarice, simulavr), link and load binary code into Atmel microcontroller flash memory (avrdude programmer) [70].

However, the Olimex boards that include AVR-CAN do not belong to Arduino project and therefore, the Arduino IDE needs to be adapted to support AT90CAN128-based boards. This adaptation had been done by [71] for the Olimex Atmega 128 Header board [72] which was a previous version of AVR-CAN board. The significant difference is that Atmega 128 Header does not include the CAN transceiver to interface this board with CAN Bus but both boards use the same microcontroller. When this adaptation was tested with AVR-CAN board and the ARDUINO IDE, the board worked without problems and only required minor details of re-configuration.

3.1.2.3 *Interface between Bridge and Back-End Modules*

The Bridge and Back-End modules are interfaced by six output pins and two input pins to interconnect all servos, including three UAS servos and two Cessna servos. Six output pins have been configured to work in PWM mode and two inputs pins have been configured to work with two ADC. The PWM output pins for UAS servos are connected to following: the pin 0C1C/PB7 is connected to UAS rudder servo, the pin 0C3B/PE4 is connected to UAS pitch servo, and the UAS roll servo is connected to PWM output pin 0C3A/PE3.

Because Cessna servos require a level of voltage and current higher than the Bridge Module can supply, H-Bridge devices are needed to interface these actuators. Each H-Bridge receives PWM signals from the Bridge module and amplifies these to supply power to its respective servo. H-Bridge one sends PWM signals to Pitch Servo SE-816A and H-Bridge two sends PWM signals to Cessna Roll Servo SA-816D. The input configuration for each H-Bridge is the following: H-Bridge one receives PWM signal from pins 0C1A/PB5 and 0C1B/PB6 through its inputs IN1 and IN2 respectively and H-Bridge two receives PWM signal from pins 0C3A/PE3 and 0C3C/PE5 corresponding to its inputs IN1 and IN2.

This last PWM output pin is shared with the H-Bridge IN1 input which supplies power to Cessna Roll Servo SA-816D and the H-Bridge IN2 input is connected to 0C3C/PE5. The Cessna pitch servo SE-816A receives the PWM signal controls from the bridge module, using the output PWM pins 0C1A/PB5 and 0C1B/PB6 through of the H-Bridge inputs IN1 and IN2 respectively.

The PWM output pin 0C3A/PE3, is shared between the UAS Roll Servo HS-985MG and the Cessna Roll Servo SA-816D. Therefore, this output pin must be configured to work with a UAS Roll Servo or a Cessna Servo. When RAFS works with a fixed-wing UAV, the system must generate a PWM signal equal to 50 Hz, and when the system works with a GAA the PWM signal must have a frequency approximately equal to 1 KHz. This configuration is implemented in software and will be analysed in Chapter four**.**

3.1.3 Back-End Module

The Back-End module is comprised of two groups of servos according to the type of aircraft which receives PWM signals generated on the Bridge Module. The first group has three servos, two HS-645MG and one HS-985MG used to work with a fixed-wing UAV. The second group has two servos, the Pitch Servo SE-816 and the Roll Servo SA-816D used to work with a GA aircraft Cessna. Both Cessna servos require two MC33926 H-Bridge drivers, one by each servo, used to

control the direction in both senses, supply power, isolate and protect electrically the bridge module against high current transients provided by servos. This section describes each element used to implement the Back-End Module.

3.1.3.1 *UAS Servo Motors Specifications*

RAFS works with two HS-645MG [73] and one HS-985MG [74] produced by HITEC which have been widely used in different types of UAS (Figure 3.8). These servos provide a satisfactory torque between 133 oz-in to 172 oz-in and speed between 0.20 sec/60º to 0.13 sec/60º respectively. HS-645MG and HS-985MG work to PWM frequency equal to 50Hz and operate between 4.8 to 6.0 Volts. Additionally, due to their low weight, 55 grams (HS-645MG) and 62 grams (HS-985MG), they can be installed to control the ailerons, elevators, rudder and throttle in small UAS.

(a) (b)

Figure 3:8 UAV Servos (a) HS-645MG [73] (b) HS-985MG [74]

3.1.3.2 *Cessna Servo Motors Specifications*

Pitch Servo SE-816A and Roll Servo SA-816D [75] were produced by the Bendix Corporation Avionics Division in 1971 to work with FCS-810 Flight Control System on board of different Cessna models like Cessna series 310, Cessna 401, Cessna 414 and Cessna 421B [76], as illustrated in Figure 3.9. FCS-810 was comprised of a 3-axis two-surface autopilot and included autopilot modes like automatic pitch trim, altitude hold, pitch command for climb or descent and preselected heading hold.

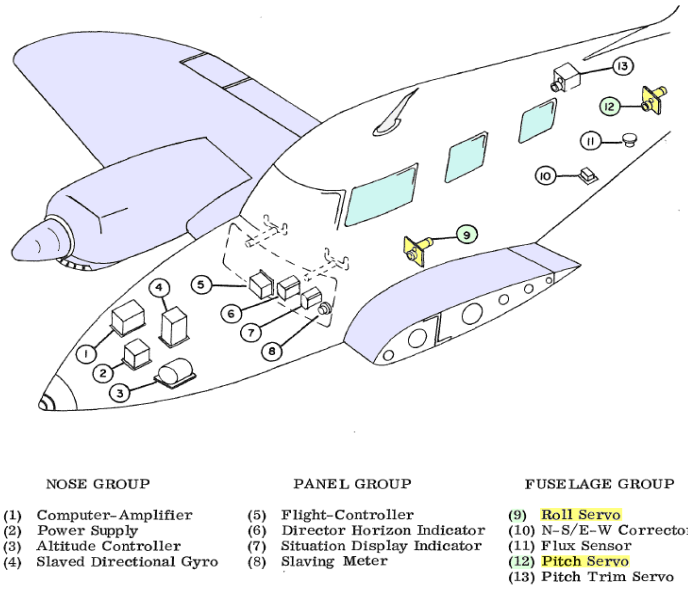
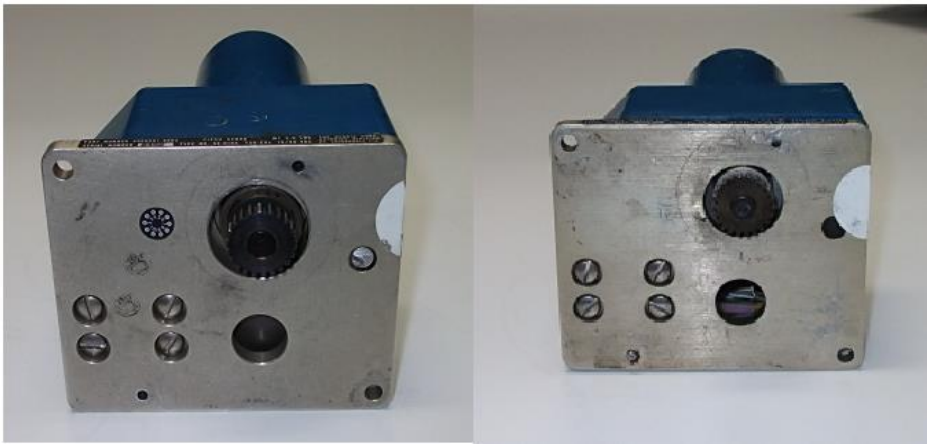


Figure 3:9 Pitch Servo SE-816A and Roll Servo SA-816D Cessna 402 [76, p. 2-2].

SE-816A is used to control the elevator and weighs 1.226 Kg whereas SA-816D is used to control the ailerons and weighs equal to 1,090 Kg. The range of operating voltage is between 14 VDC and 28VDC and current consumption is 1.0A at 14VDC and 0.5A at 28VDC. The dimensions for both servos are 4.0” x 3.3375” x 6.75”. Pitch Servo SE-816A has a high-speed gear ratio compared to Roll Servo SA-816D. Both servos have a drive motor solenoid and a mechanical clutch used when the autopilot is engaged. Figure 3.10 shows a front view of Pitch Servo SE-816A and Roll Servo SA-816D.



(a) (b)

Figure 3:10 (a) Pitch Servo SE-816A (b) Roll Servo SA-816D

3.1.3.3 *H-Bridge*

For UAS servos specifications, their voltage and current consumption are low and they can directly receive a PWM signal from Bridge module. However, the Cessna servos work a level of voltage and current much higher than the Bridge module can supply. The board Olimex AVR-CAN works between 7VDC to 12VDC whereas, the Pitch Servo SE-816A and the Roll Servo SA-816D operate between 14VDC to 24 VDC. Therefore, H-Bridges are required to interface both Cessna Servos.

The H-Bridge used in this project was the MC33926 (Figure 3.11) motor driver carrier which works at the same voltage that SE-816A and SA-816D servos. MC33926 supplies a continuous current approximately of 3A, working between 5 – 28 Volts, and may tolerate peak currents to PWM frequencies up to 20 KHz [119]. This H-Bridge offers protection against under-voltage, over-current, over-temperature and electrical isolation to Bridge module. A detailed description of the theory of operation is detailed in Appendix C. Also, the Appendix G detailed the wiring connections and the logic commands to control the H-Bridge 33926.

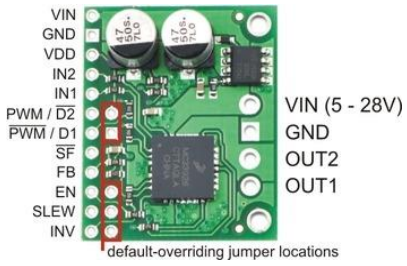


Figure 3:11 MC33926 Motor Driver Carrier [119].

3.2 SUMMARY

This Chapter has presented the architecture and design to develop the RAFS used to work for both a Fixed-Wing UAS and GA aircraft. The design is comprised of three modules, the Front-End Module based on UAS Pixhawk Autopilot, a Bridge Module based on AVR-CAN board and the Back-End Module which drives three UAS servos and two Cessna Servos. The modularity of the system is based on the CAN interface which connects the Front-End and Bridge modules. This chapter has explained the CAN Network configuration based on two physical nodes, the Front-End (Pixhawk Autopilot) node and the Bridge module node. Additionally, the hardware and software used during this project were described analysing each component and explaining the software modifications needed

Chapter 4: Reconfigurable Autopilot Flight System Implementation

RAFS software implementation was developed using two software components one in Front-End module and other in Bridge module both developed in C. Both components can be configured to work with UAV servos or with Cessna Servos, but the system cannot be configured to work with both types of servos at the same time (Figure 4.1). The first component implements the logic to adapt UAV and Cessna Servos to Front-End (Pixhawk Autopilot) module and integrates Nuttx CAN driver to transmit position commands via CAN interface. The second component runs on AVR-CAN board and implements CAN driver, the logic to generate PWM signals to UAV and Cessna servos according to position in degrees. Also, this component implements two PID control positions one for Pitch Servo SE-816A and other for Roll Servo SA-816D. This chapter explains the system configuration, the software implementation for Front-End and for Bridge modules.

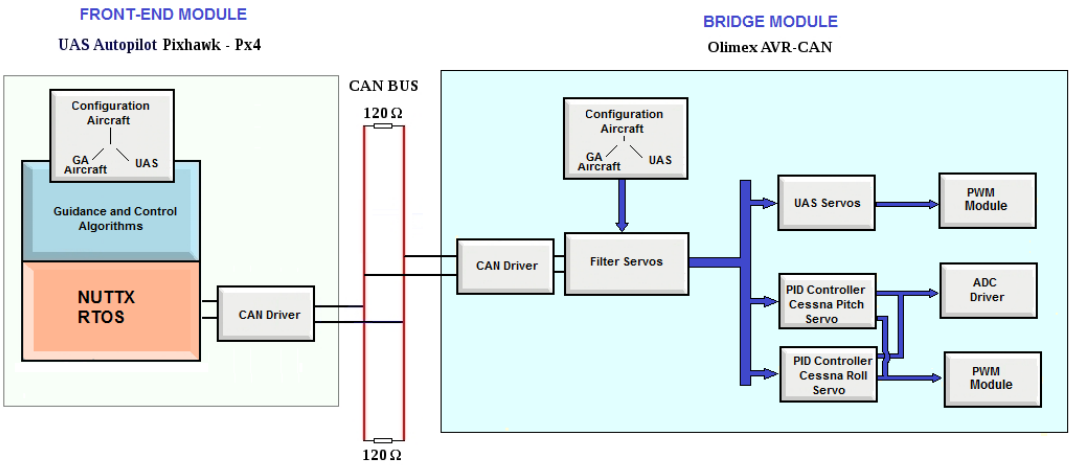


Figure 4:1 Software Architecture for Reconfigurable Autopilot Flight System (RAFS)

4.1 SYSTEM CONFIGURATION

The system configuration allows that RAFS can work with UAS Servos or Cessna Servos but not concurrently. Due to this safety criterion, the system configuration must be done during the compile time and not only during the run time. According to DO-178C [77] and MISRA [78], if an avionics system is developed for multiple applications must exist a software mechanism to deactivated code when some functions which do not require to be used for a determined application. RAFS has a condition to compile the source code, one for an UAS and other for a manned aircraft to avoid that deactivated code could have a negative effect on the active code. The

way how this is implemented is using pre-processor directives which can include or exclude lines of source code using conditionals, as illustrated in Figure 4.2. These pre-processor directives [79] are used for both sides for Pixhawk autopilot software component and the Bridge software component.

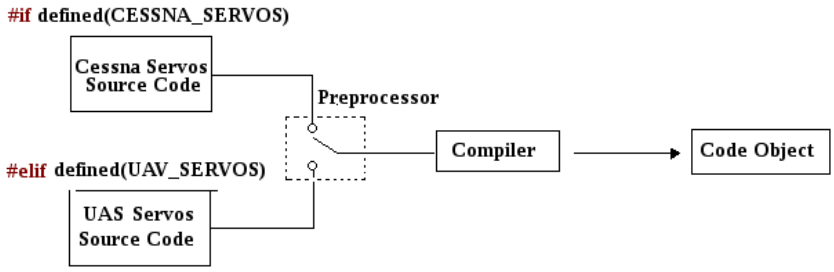


Figure 4:2 Example of source code using pre-processor directives to select the type of servos used in the system.

For instance, Figure 4.3 shows an enumeration implemented on Pixhawk autopilot and Bridge module and used to identify ID CAN nodes on the system. If the system is implemented for a Fixed-Wing UAV must define the directive UAV_SERVOS, but if the system is implemented for a Cessna Aircraft must define the directive CESSNA_SERVOS. Every time that the system must be changed from one UAS to a Cessna aircraft or from a Cessna to an UAS, the system must be compiled on Pixhawk and Bridge modules. After compiling and linking for each module of the system, the executable object code must be loaded into the Px4 and the Olimex AVR-CAN.

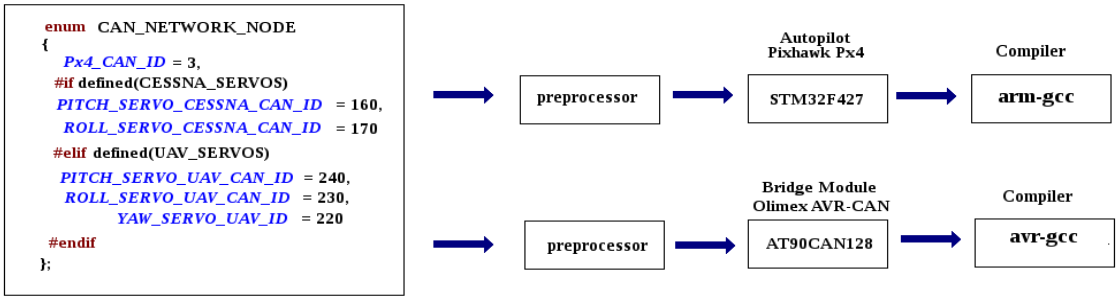


Figure 4:3 Example of source code using pre-processor directives to select the type of servos used on RAFS.

4.2 PIXHAWK SOFTWARE COMPONENTS

This software module implements a CAN connection which could be linked to autopilot guidance algorithms running on Pixhawk Autopilot. The details about the guidance algorithms implementation running on Pixhawk can be reviewed in [51] and does not correspond to this research.

4.2.1 CAN Implementation

Pixhawk Autopilot module developed CAN communication based on Nuttx RTOS. Nuttx has divided CAN driver in two levels: low and high level drivers (Figure 4.4). One is the low-level initializes and configures the microcontroller CAN registers control, status and configuration specialized to transmit and receive the CAN frames. Low-level driver sets the CAN parameters (baud rate), manages CAN interrupts, and verifies error bus status, CAN operating mode, Can filters, synchronization and timer controls [80].

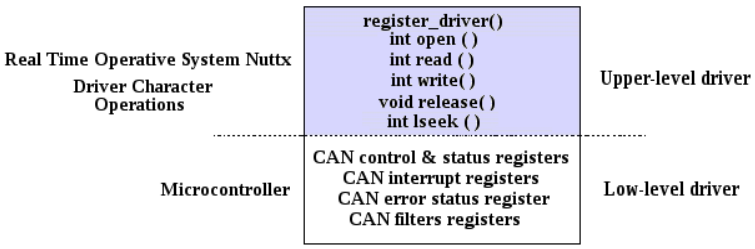


Figure 4:4 Low and upper driver levels

The upper-level driver connects the low-level driver functionalities with Nuttx RTOS tasks. All drivers in a Unix-like OS such as Nuttx are considered like character devices. It means that all devices are accessed such a file on the /dev directory using four basic operations to open, close, read, write and a function to register the driver with RTOS.

There are of step sequence to implement and execute a CAN driver in Nuttx. The first step is to initialize the driver, during second step the driver must be registered and the last step is to enable CAN interrupts (Figure 4.5). On the initialization are configured CAN port pins on the microcontroller to receive and transmit the CAN frames. The CAN_RX pin is configured as input and the CAN_TX pin is configured as output. After the initialization, the CAN driver is registered with Nuttx RTOS to link driver operations with concurrency and resources manager RTOS. Finally, application could enable CAN interrupts to send or receive messages asynchronously. After these steps CAN driver is ready to transmit or receive CAN frames over CAN bus.

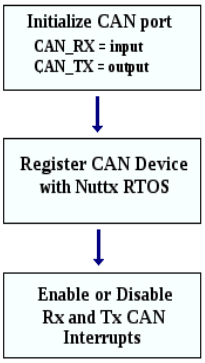


Figure 4:5 Sequence of steps to execute CAN driver on Nuttx RTOS

Using low and high level drivers is implemented CAN driver to connect Pixhawk Autopilot module to a CAN Network. Pixhawk sends positions in degrees to bridge module according to each CAN servo node identifier (Figure 4.6).

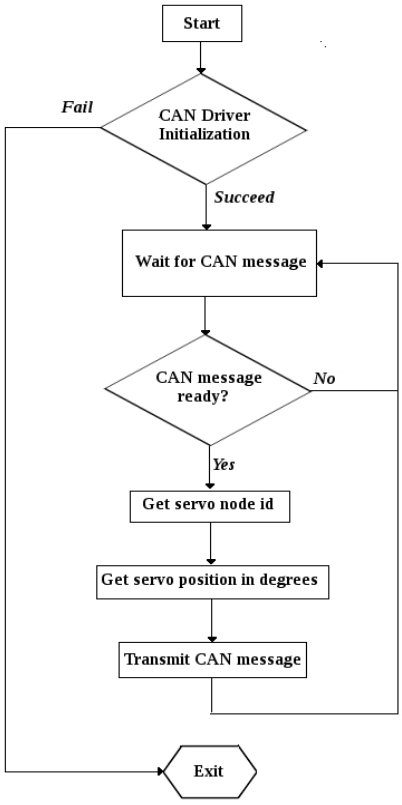


Figure 4:6 Flow char of Px4 software CAN component

4.3 BRIDGE MODULE SOFTWARE

The Bridge-module software executes two tasks to interface the Pixhawk Autopilot module with UAS servos and four tasks to interface Cessna Servos via CAN Bus. For UAS servos, these tasks include the CAN message reception and the PWM generation. For GA aircraft servos, these tasks

are to establish the CAN communication to receive servo position commands from Pixhawk, configuration and reading of two ADCs to check the GA aircraft servos current positions, configuration and execution of two standard PID position controls for Roll and Pitch GA aircraft servos and configuration and generation of PWM. This section describes the Bridge module Software configuration and implementation.

4.3.1 CAN Configuration

Using the Atmel CAN library for avr-gcc was implemented CAN protocol to work with the Bridge module. Steps to configure CAN driver consists to set the pins PD5 and PD6 corresponding to CAN pins (TXCAN and RXCAN) as output and input. After initializing PORTD, the bridge module must detect CAN activity using the RXCAN pin when the module has been connected to a CAN network. Once has been detected CAN activity on CAN bus the CAN channel is reset to baud rate equal to 250 Kbps. The Message Objects (Mob) which are structures used by AT90CAN128 microcontroller to store and handle CAN frames are cleared [64]. Finally, the CAN channel is enabled to transmit and receive messages from to CAN Bus and the Bridge module is ready to communicate with Front-End module.

4.3.2 ADC Configuration

Due to PID position control requires to know the current position for each Cessna servo, the bridge module handles two ADCs to read the voltages from two potentiometers one per each ADC. ADC configuration is based on the ADC Control and Status Register A (ADCSRA) which enables the ADC, handles the ADC interrupts and determines the sample frequency for each conversion. When the ADC driver is initialized, it is enabled but without starting conversions, the interrupts are disabled and the sample conversion frequency is set to 125 KHz. ADC Configuration is only for Cessna Servos and is not compiled for UAS servos.

4.3.3 PID Configuration

The control position for both pitch and roll Cessna servos were implemented using two Proportional-Integral-Derivative (PID) controllers. These PID controllers were developed using integer values and were programmed in a library adapted to work in Arduino. During the configuration, the values of proportional, integral and derivative constants are set according to a previous tunning got at the laboratory (Section 5.2.2). Only the PID configuration is used to work with GA aircraft servos and not for UAS servos.

4.3.4 PWM Configuration

For both types of unmanned and manned aircraft servos a Pulse Width Modulated (PWM) is applied. However, the frequency used to work in UAS servos is lower than Cessna servos.

According to manufacturer, the PWM frequency required to work with UAS servos is 50 Hz whereas the PWM frequency used to work Cessna servos is 1 KHz. Furthermore, UAS servos only require one a PWM signal to control the position, but the Cessna servos require two PWM signals. These two PWM signals control the sense of rotation when the servo goes forward or goes reverse and are applied to H-Bridge inputs (see Appendix C). The program logic must avoid that two PWM signals be generated at the same time to H-Bridge inputs to prevent a short circuit.

The bridge-module PWM configuration uses two 16-bit timers (Timer1 and Timer2) to work 16-bit PWM signals. The AT90CAN128 microcontroller has four PWM modes of operation: normal mode, fast PWM mode, phase correct PWM mode and phase and frequency correct PWM mode [64]. The phase and frequency correct PWM mode was used to work during this project.

4.3.4.1 *Phase and frequency correct PWM mode*

This mode is ideal to work motor control applications due to a dual-slope operation as shown in Figure 4.7 which provides a lower maximum operation frequency compared to others three PWM modes. To generate a PWM signal using this mode, five 16-bit registers must be configured depending which timer (Timer 1 or Timer 2) is implemented. The necessary PWM registers are: Timer/Counter Control Registers A and B (TCCRnA, TCCRnB), Timer Counter (TCNTn), Output Compare Register (OCRnx) and Input Capture Register (ICRn). To following, an example explains the phase and frequency mode.

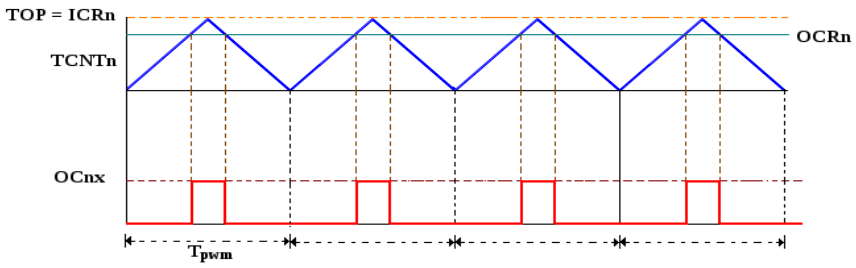


Figure 4:7 Phase and frequency correct PWM mode. Adapted from [64].

The blue triangular signal represents TCNTn register (1 or 3), the orange dotted line represents the ICRn register, the green line represents the OCRn register and the red signal represents the PWM signal on the OCnx pin. When the Time Counter (TCNTn) matches the ICRn value, this begins to count in countdown until zero completing a PWM period. While Time Counter (TCNTn) value is bigger than OCRn value the output pwm signal on OCnx pin is high. When Time Counter value is lower than OCRn, the output OCnx is equal to zero.

The equation (4.1) allows setting the PWM frequency where f_{clk_IO} is the system clock (16 Mhz), N represents the prescaler divider (1, 8, 64, 256 or 1024) and TOP represents the ICRn register value.

$$f\_pwm = \frac{fclk_{\_i/o}}{2*N*TOP} \quad (4.1)$$

To generate a PWM frequency equal to 50Hz using UAS servos, the chosen values of each variable (equation 4.1) are N equal to 8 and TOP equal to 20000. On the other hand, the PWM frequency used to work Cessna Servos is 976.5625Hz using a prescaler factor N equal to 8 and TOP equal to 1024. The prescaler factor must be configured on the Timer/Counter Control Register B (TCCRnB, n:1,3) and the TOP value must be loaded into the ICRn register [64].

4.3.5 Main Loop

For both UAS servos and Cessna Servos the Bridge module waits until a command position is transmitted from the Pixhawk autopilot via a 4-byte CAN standard frame. After a command position has been received, the Bridge module identifies the CAN node id to which Pixhawk autopilot has sent a command. Each CAN node id is matched to a servo motor according the software configuration, if the Bridge module has been compiled to work with the UAS servos or Cessna servos, as described in Figure 3.6, CAN Network Topology.

4.3.5.1 *Configuration Fixed-Wing UAS*

If the system has been configured to work with an UAS and once the Bridge module has identified what servo needs to change its position; the module gets the desired position in degrees stored at the first data byte of the CAN frame. Using the desired position is calculated the PWM duty cycle according to equation 4.2:

$$int\ uav\_pos\_microseconds = PWM\_DUTY\_CYCLE\_NEUTRAL\_POS\_USEC + 10 * pos\_degree \quad (4.2)$$

Where constant "PWM_MIN__DUTY_CYCLE_USEC" equal to 1500us is the PWM duty cycle for a UAS Servo neutral position and the parameter "pos_degree" is the desired position in degrees. This equation was obtained according to manufacturer datasheet [81], as shown in Figure 4.8.

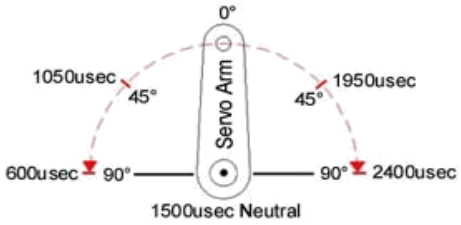


Figure 4:8 Relation between PWM duty cycle and servo position [81].

The result of equation 4.2 is a PWM duty cycle in microseconds which is equivalent to UAS Servo position. The result *uav_pos_microseconds* is loaded into Output Compare Register (OCRn) which adjusts the PWM duty cycle in the microcontroller AT90CAN128. Each UAS servo has its own OCRn register where it is possible to change the PWM cycle duty, as shown by Figure 4.9. When the command position has been executed the function returns to main function and the system again waits for a new CAN message from the Pixhawk Autopilot.

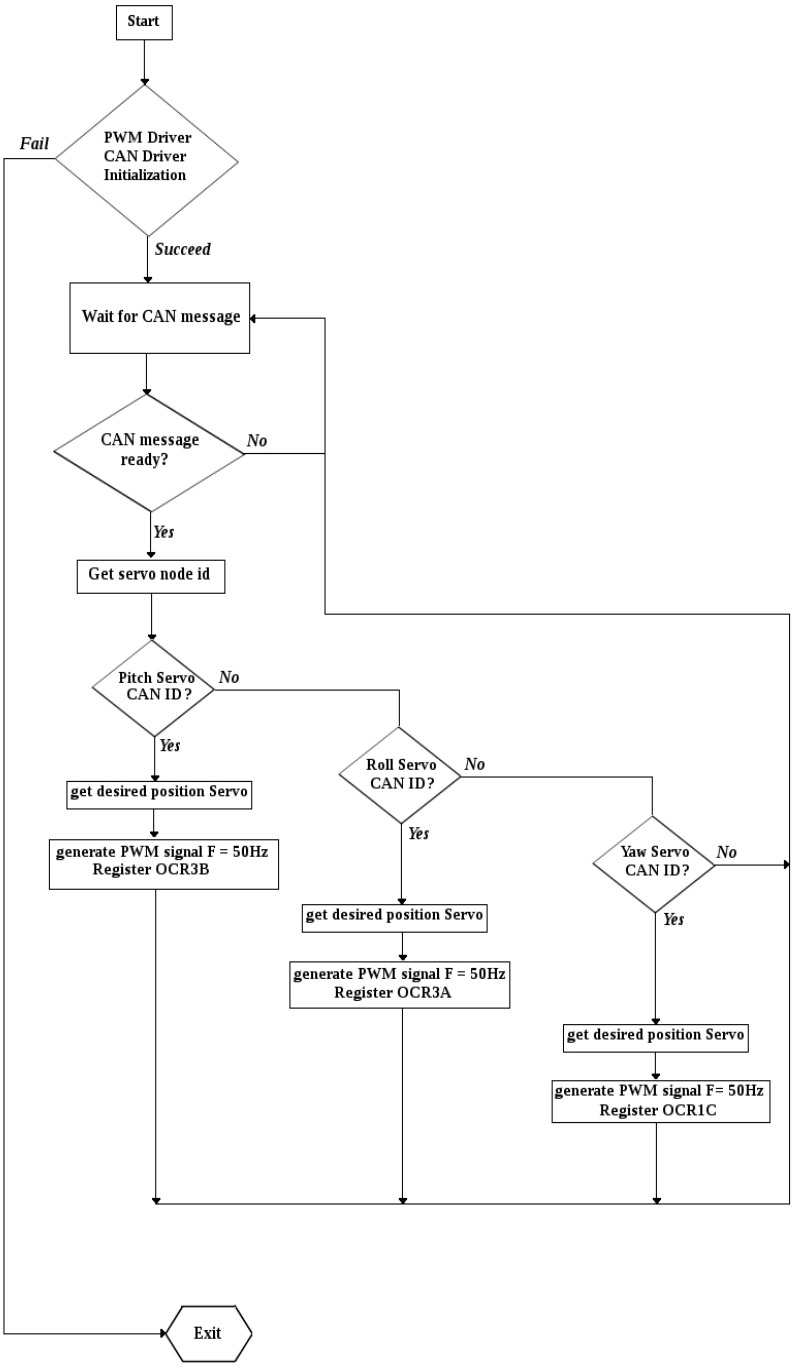


Figure 4:9: Flow char of Bridge Module compiled for UAS servos.

4.3.5.2 *Configuration General Aviation Aircraft*

If the system has been configured to work with a GA aircraft, there are two important differences respect to Fixed-wing UAS configuration that must be implemented in software. One difference is given by the number of PWM signals that each servo received and the second difference is that Cessna Servos requires the control positions. During the main loop the program initializes the PWM, CAN, ADC and PID drivers and after waits that a CAN message be transmitted from Pixhawk Autopilot module. When a CAN frame is received the Bridge module recognizes if the

message was send to a Cessna Pitch or Roll Servo according to a CAN node identifier. Once, the system has recognized the type of servo applies a PID controller to control the position (Figure 4.10).

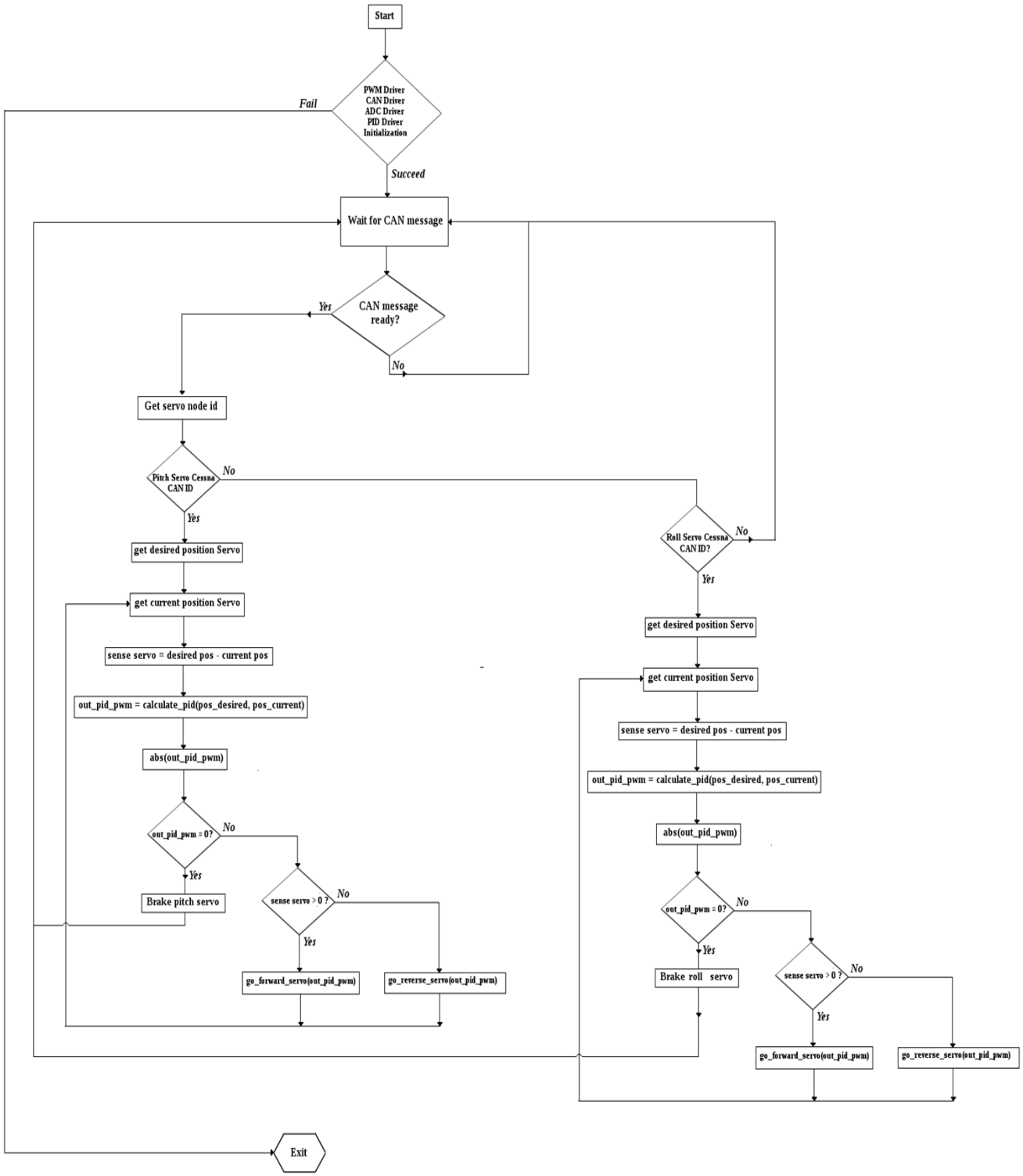
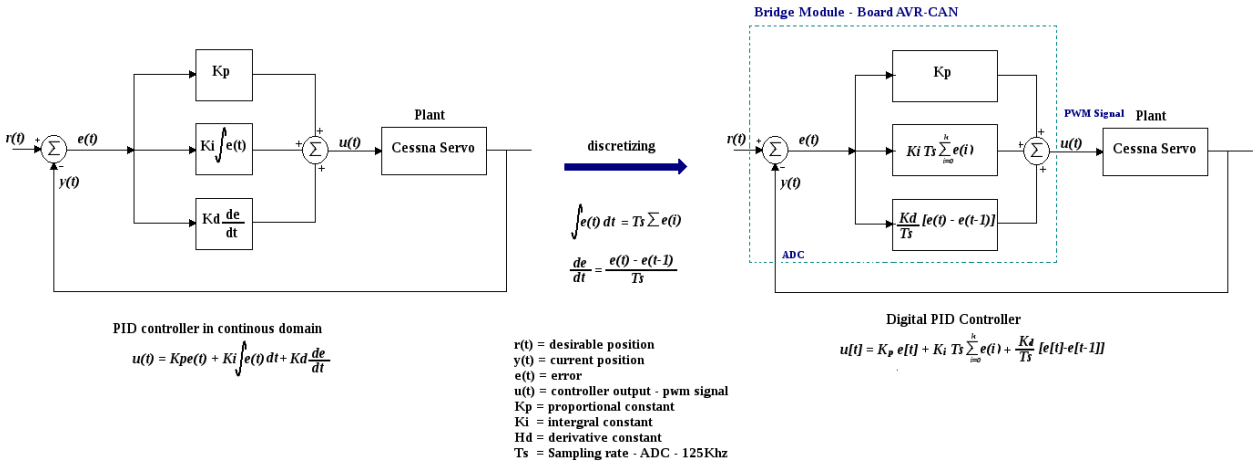


Figure 4:10 Flow char of Bridge Module compiled for Cessna servos

4.3.5.2.1 PID Position Control

To control the servo positions for both Cessna Roll and Pitch Servos was necessary to implement a standard PID digital controller. Often, aircraft control surfaces (ailerons and elevators) may be subjected to atmospheric conditions such as high speed wind or rain during a flight. As a result, actuators could experiment variations in their positions that must continuously be corrected by the Autopilot System. Due to this reason, Bendix Corporation suggests in the servo specifications that control positions must be implemented for the pitch servo SE-816A and the roll servo SA-816D when are installed on aircraft. Although to develop a new algorithms or PID technique is beyond of scope of this project some PID concepts will be mentioned to explain how the PID position control was implemented. A wide and detailed literature about PID control can be found in [82, 83].



(a) (b)

Figure 4:11 PID Controller implemented on Bridge Module (a) Analogue PID (b) Digital PID

Figure 4.11 shows a standard PID controller in its analogue version and its digital version. As well known to implement a digital control system, the analogue model must be discretised to transform in an algorithm. The digital model implemented is based on the analogue PID controller [85] defined by the continuous domain equation (4.3).

$$u(t) = Kpe(t) + Ki \int e(t) + Kd\frac{de}{dt} \qquad (4.3)$$

To discretise the equation (4.3), the integral and derivative terms are approximated using (4.4) and (4.5) according to [84]:

$$\int_0^t e(t)dt \approx T_s \sum_0^t e(i) \qquad (4.4)$$

$$\frac{de}{dt} = \frac{e(t)-e(t-1)}{T_s} \qquad (4.5)$$

Replacing (4.4) and (4.5) in (4.3) is obtained a satisfactory discretization of PID controller (4.6).

$$u(t) = K_p e(t) + K_i T_s \sum_{i=0}^{k} e(i) + \frac{K_d}{T_s}[e(t) - e(t-1)] \qquad (4.6)$$

Using the difference equation (4.6) is implemented the digital PID controller algorithm. The PID algorithm receives the desired and current positions, as illustrated in Figure 4.12. The ADC reads the voltage variations of a potentiometer which corresponding to shaft position. After, the current error is determined calculating the difference between the desired and current positions. The integral part is determined calculating the summation of previous errors and derivative part is obtained calculating difference between the current and previous errors. Finally, the PID output is estimated multiplying each PID term per its respective constant and adding the three parts. The PID output is loaded into Output Compare Register (OCRnx), as illustrated in flow chart (Figure 4.12).

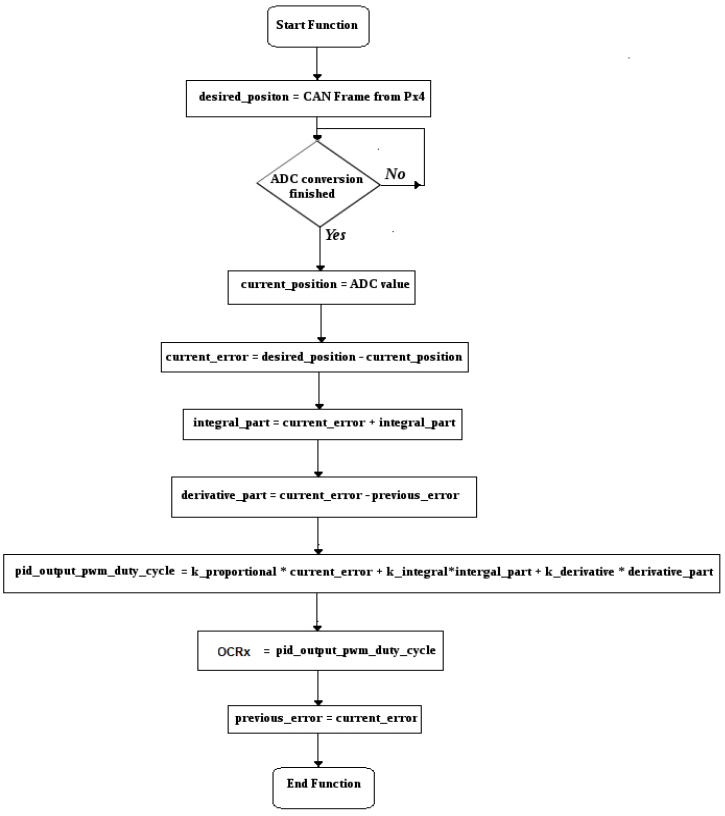


Figure 4:12: Flow chart of PID Controller algorithm

Once obtained the PID-controller result, the program verifies if the result is equal to zero indicating that the servo has reached the desired position. However, if the desired position has not

been reached, it is defined the servo sense of rotation. When the current error is positive the servo goes forward but if the current is negative the servo goes reverse trying always decreasing the error (Figure 4.10). The PWM duty cycle is proportional to current error. It means that when the current error is decreased, the duty cycle is reduced as well. Finally, a PWM signal is transmitted to H-Bridge according to sense of rotation. If the servo goes forward the signal is sent to H-Bridge input 1 but if the servo goes reverse the signal is sent to H-Bridge input 2. Two PWM signals will never be applied at the same time to both H-Bridge inputs. When the servo has reached the target position the program stops the servo and returns to main loop to wait a new command from Pixhawk autopilot.

4.4 SUMMARY

This Chapter has presented the implementation of RAFS explaining the system configuration, the logic to implement CAN connection on Front-End (Pixhawk Autopilot) module and Bridge module and the logic to send PWM signal to UAS and Cessna servos. The system configuration has been implemented to work during compiling time instead of run time due to safety criteria according to DO-178 and MISRA standard coding. Also, the logic related to link the CAN driver to Pixhawk Autopilot is explained taking advantage of driver implementation used in Nuttx RTOS. Additionally, the PWM signal generation and PID control have been explained and detailed showing the differences when RAFS is configured to work with a UAS and when is configured to work with a GA aircraft.

Chapter 5: Validation and Results

This chapter describes the tests used to validate the Reconfigurable Autopilot Flight System for General Aviation and UAS. The validation and results of this research are based on three tests. The first test verifies the communication between the Front-End module and Bridge Module using a CAN sniffer which checks the position commands sent from Pixhawk Autopilot. The second test verifies the communication between the Bridge Module and UAS Servos checking the PWM signal generated in the Bridge module. The third test verifies the PID position control for GAA Servos evaluating the position command transmitted from Pixhawk Autopilot and the observed position for Pitch and Roll Servo. Also, the step response the data obtained for both Cessna servos are showed in detail.

5.1 FRONT-END MODULE AND BRIDGE MODULE COMMUNICATION

The communication was tested using a third CAN node which interconnects a laptop with the CAN Network to verify the CAN transmission between the UAS autopilot Pixhawk and the Bridge Module, as illustrated in Figure 5.1. The adapter used to connect the laptop to CAN bus was the CANUSB Lawicel [86] and the open-source driver was jCAN which included a CAN sniffer. This sniffer detected the messages transmitted from Pixhawk to Bridge Module.

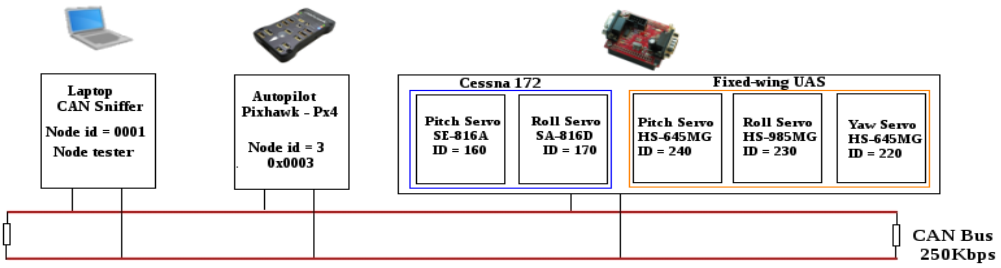


Figure 5:1 CAN Sniffer integrated into the CAN Network topology.

The test was developed using the following procedure. First, the laptop is connected to CAN Network using a CAN node identifier equal to 0001. After, the Pixhawk autopilot transmits the position command on the first data byte of the CAN frame according to each CAN servo identifier (Figure 5.2 and 5.3). The bridge module receives the CAN messages and according the CAN node id distributes the command position to each servo. As a result, the observed position for each servo is compared with the position in degrees transmitted by the Pixhawk UAS Autopilot. This procedure was done when the source code was compiled for a Fixed-Wing UAS and when the source code was compiled for GA Aircraft. For instance, if the RAFS was configured to work with a Fixed-Wing UAS only position commands for UAS servos should be transmitted to Bridge Module. When RAFS was

configured to work with a GA Aircraft only position commands for Cessna Servos should be transmitted to Bridge Module.

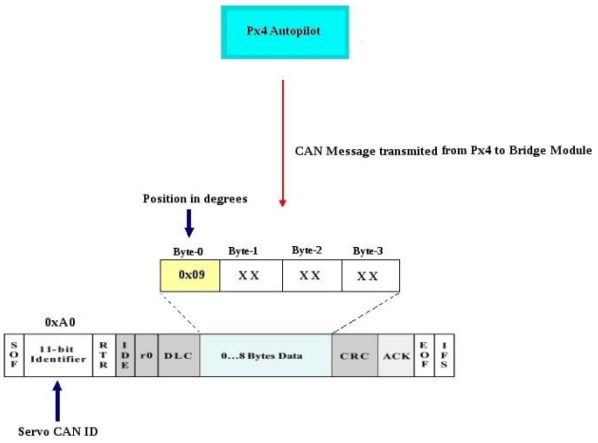


Figure 5:2 CAN message transmitted from Px4 to Bridge Module

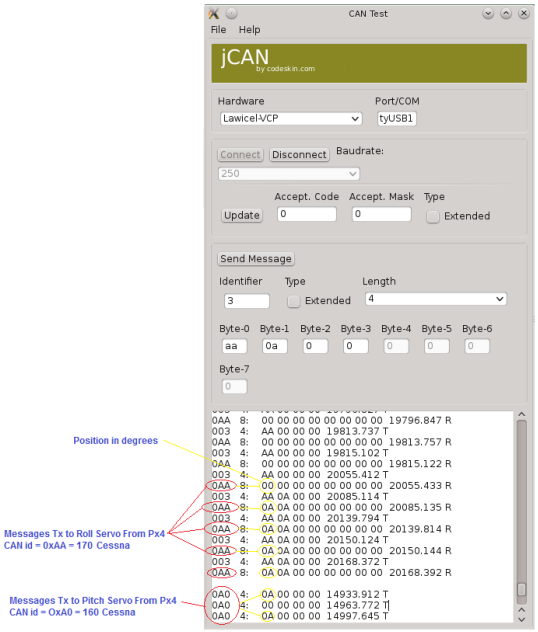


Figure 5:3 CAN messages transmitted from Px4 to Bridge Module detected by jCAN Sniffer

The outcome of the CAN connection between the Front-End module and the Bridge module shows that each CAN frame spend maximum 20ms to be transmitted for both system configurations GA Aircraft and Fixed-Wing UAS. The speed of data transfer used to connect both modules was of 250 KHz using a Standard CAN 11-Bit identifier. Loss of CAN messages was not detected during the tests. However, sometimes the CAN Sniffer detected CAN Bus error events at the beginning of the connection when multiple position commands were sent from Front-End module.

5.2 BRIDGE MODULE AND BACK-END MODULE COMMUNICATION

5.2.1 UAS Servos

The outcome of the CAN connection between the Front-End module and the Bridge module shows that each CAN frame spend maximum 20ms to be transmitted for both system configurations GA Aircraft and Fixed-Wing UAS. The speed of data transfer used to connect both modules was of 250 KHz using a Standard CAN 11-Bit identifier. Loss of CAN messages was not detected during the tests. However, sometimes the CAN Sniffer detected CAN Bus error events at the beginning of the connection when multiple position commands were sent from Front-End module.

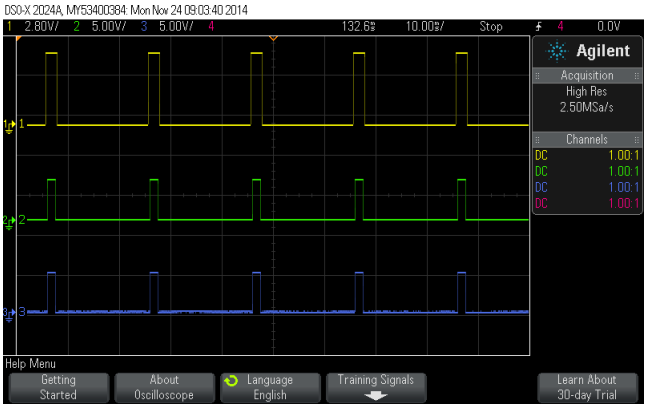


Figure 5:4 PWM signals transmitted from Bridge Module to UAS servos

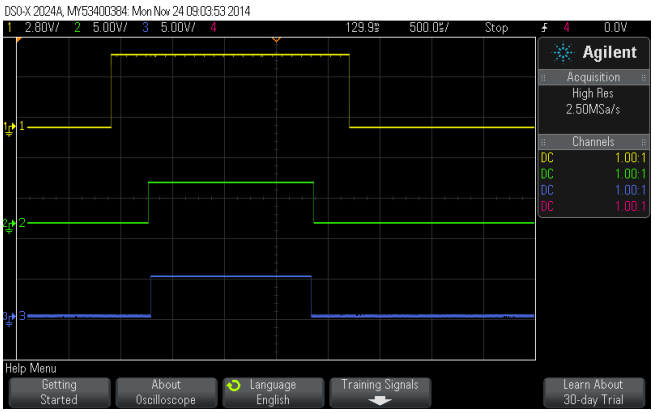


Figure 5:5 Zoom of PWM signals for UAS servos where yellow signal corresponds to Rudder Servo (90 degrees), green signal corresponds to Roll Servo (10 degrees) and blue signal corresponds to Pitch Servo (5 degrees)

5.2.2 Cessna Servos

To implement the position control for Cessna servos we used a standard PID controller for each servo. The feedback position was developed using a potentiometer where the voltage variations were read by 10-bit ADC on the Bridge Module. Using a reference Voltage equal to 5 Volts and ADC resolution of 10 bits where the ratio between volts and bits is equal to 4.8874 (Equation 5.1)

$$Q = \frac{5000\ mVolts}{1023} = 4.8874 mV/bit \quad (5.1)$$

The tunning for the PID control position was done experimentally changing the values of the PID constants and observing the number of degrees that one of each shaft servo spun. The desired position send from Pixhawk Autopilot was compared with the observed position. The data position was acquired using a disk with the position in degrees mounted on the servo shaft and sending PWM signals to H-Bridge driver in reverse and forward direction (Figure 5.6).

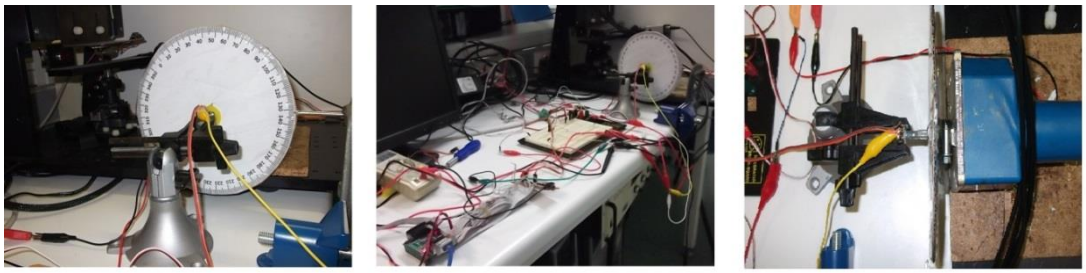


Figure 5:6 Cessna Pitch Servo SE-816A Experimental test to tune the position controller

Table 5.1 shows the PID constants of position controller for both Pitch and Roll Servos tunned experimentally in the laboratory.

	K_P	K_I	K_D
Pitch Servo (SE-816A)	25	1	25
Roll Servo (SA-816D)	35	1	20

Table 5:1 Constants for PID position Controller for Pitch and Roll Servos

The step response for Pitch Servo SE-816A shows that there is not overshoot and the rising time is approximately of 20*us*, as illustrated in Figure 5.7 (a). Similar to Pitch Servo response, the step response for Roll Servo SA-816D does not show overshoot and the rising time is 20*us* as well (Figure 5.7 (b)). Both step responses for Pitch and Roll Servos are in purple colour, however the scale for the step response is major than the input signal (blue and green signal) due to the servos work on a voltage equal to 28 Volts whereas the input signal has a voltage equal to 5 Volts. Both step responses indicate that each PID controller is stable without oscillations and with a short response time.

Pitch Servo (SE-816A) Roll Servo (SA-816D)

Figure 5:7 Step response of angular position for Cessna servos.

Tables 5.2 and 5.3 show the results obtained at the laboratory and compare the command position transmitted from Pixhawk Autopilot and the position measured experimentally. The third column for both tables show the feedback voltage measured on the potentiometer. Both servos have a similar behaviour, but the Roll Servo requires a greater value for proportional constant than Pitch servo.

Position sent	Position observed	Feedback Voltage (Volts)
1	1	2.50
5	5	2.60
10	10	2.70
15	15	2.80
20	20	2.90
25	25.5	3.050
30	31	3.150
35	36	3.3
40	41.5	3.4

Table 5:2 Comparison between the Degree Sent from Pixhawk Autopilot and the Degree Observed for Pitch Servo SE-816A

Degree sent	Degree observe	ADC Input Value (Volts)
1	1	2.54
5	5	2.62
10	10.0	2.73
15	15	2.83
20	20	2.92
25	25	3.03
30	30	3.12
35	35.4	3.22
40	40.4	3.32

Table 5:3 Comparisons between the Degree Sent from Pixhawk Autopilot and the Degree Observed for Roll Servo SA-816D

5.3 SUMMARY

Three tests were done to evaluate RAFS. The first test analysed and verified the data transmission via CAN Bus between the UAS autopilot and Bridge module using a CAN sniffer. This test checked the communication of position command generated from the UAS autopilot when the system was configured to work with a UAS Aircraft or for a GA Aircraft. The second test verified the PWM signal generation on the Bridge module for UAS which must be related to position command transmitted from Pixhawk. At the same time, the angular positions were verified for UAS servos according to the observed position for each servo. Finally, the third test evaluated the operation of PID position controller analysing the step response for both Cessna Servos and comparing the transmitted position from Pixhawk Autopilot and observed position for Roll and Pitch Servos.

Chapter 6: Conclusions

The capability increase in hardware and software technology has made UAS autopilots more sophisticated and powerful, and in some instances more capable than Autopilot Systems used in General Aviation. The main of this project has been to research, design, develop, implement and test at the laboratory (hardware in the loop) an Autopilot Flight System which can be configured and integrated to work for both an UAS and GA Aircraft using open-source hardware and software components. To achieve this aim this research assessed different interfaces and autopilot designs used for GA aircraft and UAS.

During this research the following considerations about the design and integration of Autopilot Systems in GA and UAS were devised:

- Many Avionics autopilots for GA Aircraft do not allow the integration to others sensors and actuators developed by different manufacturers, because they develop applications using property interfaces which avoid integrating different avionics modules.

- Manufacturers implement they own version of standard protocols to interface their own avionics modules. However, although the implementation of these standard protocols should increase the modularity between different manufacturers, the type of messages that are sent between components is only known by the autopilot manufacturer. As a result, if new avionics components were integrated to autopilot, these could not understand the messages transmitted by autopilot system because they have not been developed by the same manufacturer.

- The GA Aircraft manufacturers and UAS manufacturers use different type of interfaces to communicate sensors and servos with avionics systems. GA Aircraft manufacturers use high-level protocol because they bring features such as immunity again vibration and white noise, high level of error detection and the data transmission can be sent major distances. Whereas UAS autopilots design use low-level protocols to interface their I/O peripherals, but these protocols do not bring optimal error detection and they do not cover the same distance than GA protocols.

- New autopilot architecture for UAS tends to make more modular I/O to isolate the rest of peripherals and using dedicated hardware to run hard-processing algorithms. However,

these UAS autopilot architectures do not use high level protocol to integrate their sensors and actuators. A couple research projects have used CAN interface to integrate different UAS systems bringing the opportunity to integrate GA Aircraft modules with UAS autopilots.

- However, neither project has tried to develop an Autopilot System for both manned and unmanned aircraft.

Given the importance of open-source hardware and software projects for UAS autopilots and their potential application for General Aviation is recommendable that open-source projects implement strategies that follow the Safety Standards for Hardware and Software Development. This would make possible to create applications for airborne industry achieving their safety requirements. However, because of open-source projects are open multiple developers to interact and create or improve the software and hardware, the tests to check the applications are not executed with the requirements that airborne industry demands. As a result, open-source projects do not bring warranties in their applications and can presents problems when are used to implement new applications for manned aircraft.

This research concluded that the modification of an Autopilot used on GA to be implemented in a UAS system is a not viable option because of features such as dimensions, weight and lack of modularity with sensors and actuators among others prevents installation of GA Autopilot on-board of Fixed-Wing UAS. Furthermore, GA autopilots do not offer communication with a ground station which is essential for UAS. GA manufacturers do not allow software and hardware modifications. Whereas open-source UAS autopilots allow modifying and integrating several hardware modules including CAN avionics modules and the physical characteristics such size and weight are compatible for GA Aircraft. Therefore, the main contribution of this thesis has been the design, development and implementation of Autopilot for both UAS and GAA aircraft based on UAS autopilot via CAN interface and implementing a Bridge Module to manage the group of servos.

6.1 FUTURE WORK

The results of this research have provided new avenues for future work which includes the following considerations:

- The implementation of higher level protocols based on CAN for UAS autopilots so that commercial or general aviation applications can be used in UAS. For instance, the

implementation of protocols such as CANAerospace or ARINC 825 which will provide modularity and compatibility with GA Aircraft.

- The creation of strategies for open-source UAS autopilots projects to follow Avionics Standards during the process of hardware and software development. This would bring major possibilities to increase the safety requirements and therefore the open-source UAS autopilots could be certified for use on GA aircraft.

- The integration of additional Human Machine Interfaces used in cockpit, group of sensors and actuators with an UAS Autopilot and the execution of experiments where all the UAS autopilot system and GA components could be used on-board on manned aircraft and vice versa.

- The implementation of a Real Time Operative System (RTOS) on the Bridge Module to improve the task and process management and therefore increasing the capabilities of the system allowing the connection with multiple sensors and avionics modules working at the same time with a UAS Autopilot on-board of UAS or GA Aircraft.

- A detailed analysis about the characterisation of the Pitch Servo SE-816A and Roll Servo SA-816D to find the transfer function for both servos to achieve a better performance on the PID controller.

Appendix A: **Safety Standards for Hardware and Software Development in Avionics industry**

In the avionics industry safety is the most important aspect in any phase of the design, development, implementation, production and operation of an aircraft. The fact that human lives could be affected in their physical wellbeing during a flight demands that aviation companies must follow strict rules and controls to ensure the safety. These rules have been created, classified and documented in international standards by the airborne industry and governmental organizations such as EUROCAE in Europe and RTCA in USA. The aim of aviation standards is to set guidelines to increase the safety detecting failures in development processes. Safety standards must be followed by avionics manufacturers to certify hardware and software processes. It means that an open-source hardware and software component such as an UAS autopilot should follow these rules to get an airborne certification and therefore, it could be used on-board of a GA Aircraft. This appendix introduces an overview of software and hardware standards widely used in airborne applications or aviation equipment for both manned and unmanned aircraft focusing in the development of an Autopilot Flight System (AFS).

The civil aviation standards introduced in this appendix have been published by the Radio Technical Commission for Aeronautics (RTCA). This organization was founded in 1935 in Washington DC USA, and is in charged to develop guidelines for the airborne industry. The RTCA is comprised of the most prestigious avionic manufacturers, and airborne international government entities around the world which publish and update these standards. The RTCA standards are used by the Federal Aviation Administration (FAA) to determine the necessary policies to develop hardware and software and implement new airborne technologies.

The standards introduced in this appendix have been classified in four types. The first type is the standard RTCA DO-325 "*Minimum Operation Performance Standards (MOPS) for Automatic Flight Guidance and Control Systems Equipment*" used to develop AFS. The second type gathers the civil standards for UAS RTCA DO-304 "*Considerations for Unmanned Aircraft Systems (UAS)*" and RTCA DO-344 "*Operational and Functional Requirements and Safety Objectives for UAS*". The third type gathers the software standards RTCA DO-178 "*Software Considerations for Airborne Systems and Equipment Certification*", RTCA DO-332 "*Object-Oriented Technology and Related Techniques*", RTCA DO-330 "*Software Tool Qualification Considerations*" and the coding standards. Finally, the hardware RTCA standards DO-254 "*Design Assurance Guidance for Airborne Electronic Hardware*" is presented. Figure A:1 shows the standards related to develop and implement an AFS.

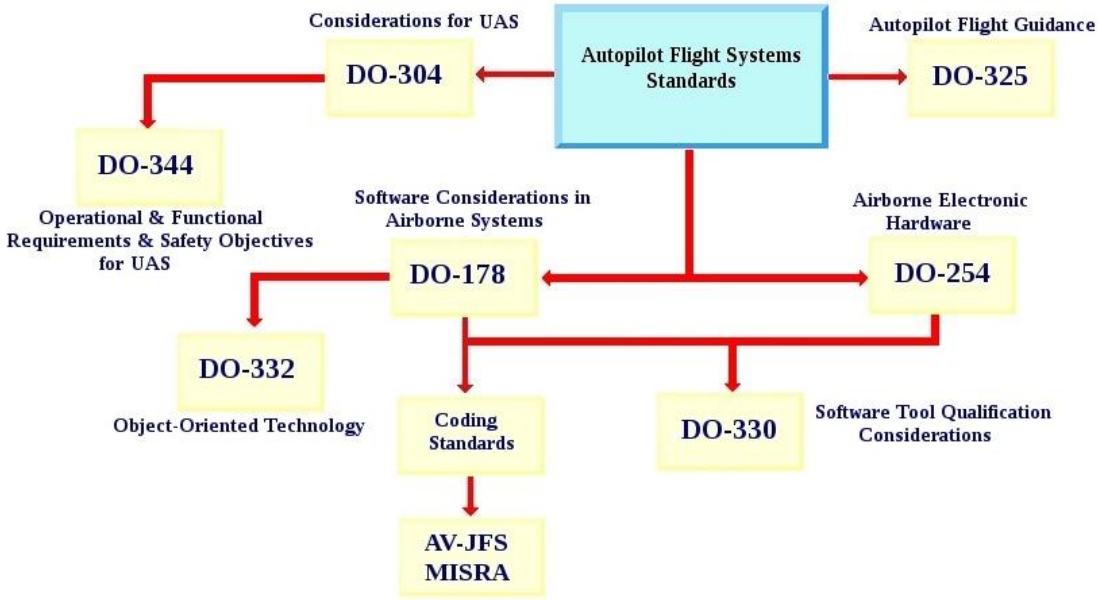


Figure A:1 RTCA Standards for Software and Hardware Development for a AFS..

A.1 RTCA DO-325. Minimum Operation Performance Standards (MOPS) for Automatic Flight guidance and Control Systems and Equipment

The standard DO-325 defines the different characteristics that an AFS must have when is installed in a civil aircraft. Additionally, DO-325 defines the main function an autonomous flight system, which is to support the piloting of an aircraft without human assistance. DO-325 specifies the functions and modes that an autopilot must have. The vertical modes include the Pitch Attitude Hold Mode, the Vertical Speed Hold Mode, the Flight Path Angle Mode, the Altitude Hold Mode and Flight Level Change Mode. The standard establishes lateral path control functions such as Roll Attitude Hold Mode, Heading Mode, Track Angle Acquire Hold, and Localizer Mode. Also, some autothrottle functions are suggested, such as Airspeed control mode, thrust control, thrust reduction and take-off or go-around [87].

DO-325 mentions in a generic way the hardware and software that should be included an AFS. Also, the standard suggests that the AFS should be tested under different weather conditions according to standard RTCA DO-160G [88]. Environmental tests include temperature variation, humidity exposure, crash-safety shocks, vibration, magnetic effect, radio frequency susceptibility and icing that servos and sensors could suffer. The system must emit alerts in case the autopilot identifies a failure during a flight. This standard specifies the servo functions indicating the maximum limit for the pitch and roll axes, the maximum servo force, and the maximum load condition indicating a

minimum strength load of 2.5 times. Finally, DO-325 presents guidelines about the autopilot installation inside an aircraft and the steps required to configure the mechanical and electrical components and the verification of the correct installation into the cockpit.

A.2 Civil Standards for UAS

Unlike general aviation manufacturers which must accomplish demanding rules and guidelines to develop avionics products, the UAS has lacked of regulations and policies that may coordinate the production of Unmanned Aerial Vehicles (UAV). Nonetheless, due to increasing and rapid technological advance in UAS, the international airborne authorities have begun to require that UAV manufacturers follow some standards used in the civil aviation. To following the UAS-related standards RTCA DO-304 and RTCA DO-344 are analysed.

A.2.1 RTCA DO-304 Considerations for UAS

The coming of UAS into civil aviation has made airborne governmental organizations and airborne companies include new guidelines which allow that UAVs can interact in National Aerial Space (NAS). The FAA and RTCA created DO-304 [89] to clarify procedures, detailed concepts and regulations about UAS and its interaction with civil and commercial aircraft and people. DO-304 distinguishes four segments related to UAS operations, one is the aircraft segment, second is the control segment, third is the communication segment and the NAS segment as shown in Figure A.2.

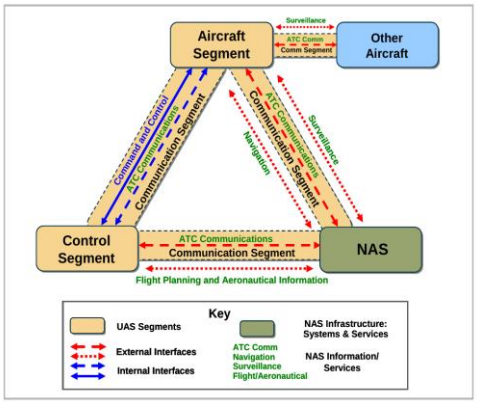


Figure A:2 UAS Segments according RTCA DO-304 [89].

A.2.1.1 Aircraft Segment

The aircraft segment considers the hardware and software elements and includes aspects such as the flight management control, the flight plan management and the control link. The flight management includes the flight guidance, flight recovery, traffic avoidance, and weather avoidance. The standard suggests that software and hardware must involve safety and security procedures related to flight guidance like flight recovery, aircraft health and flight status information. Second, the aircraft segment analyses the flight plan management which should determine the position and guidance.

Furthermore, DO-304 considers the control link in charge to receive and transmit the orders from and to the ground station, including the Air Traffic Control (ATC) communications, antennas and sensors. The received and transmitted information must report the aircraft health, status and telemetry.

A.2.1.2 Control Segment

The control segment refers to the UAS crew who remotely control the aircraft segment and the functionality, facilities and equipment required to command the unmanned aircraft. This segment includes mainly subsystems such as the flight control operation, control link equipment, communications link equipment and flight planning. For flight control operation must include the sent and received commands from ground station or from aircraft segment and the monitor of aircraft health and status. Communication link equipment refers to equipment used to transmit voice or data to traffic controllers or other aircraft from ground station.

A.2.1.3 Communication Segment

Communication segment refers to internal and external interfaces used to support the communications systems. The internal interfaces supply the connectivity necessary to communicate the aircraft and control segments using the different displays and human machines interface inside the unmanned aerial vehicle. With respect to the external interfaces, these manage the UAS interaction within the National Aerial Space (NAS), analysing aspects such as the ATC communications, surveillance, navigation, flight planning, the physical sensor inputs and the aeronautical information services.

A.2.1.4 National Airspace System Segment

This segment discusses the way UAS must be integrated into NAS. The National Airspace System is known as the U.S. airspace Network and includes the air navigation facilities, the equipment involved in the avionics operations, airports or landing areas, rules, regulations, and technical information used in the aerospace industry. Some other important considerations are:

- All software, hardware and firmware must follow the criteria of the FAA outlined in the standards RTCA-DO-178 [90] and RTCA-DO-254 [91] used by the avionics manufacturer.
- The UAS pilots will need to have a certification to pilot an unmanned aircraft.
- When an UAS emits a failure system, the unmanned aircraft must guarantee the emergency procedures to land the aircraft and in those cases where the failure cannot be fixed the operator has the option to finish the flight. Furthermore, the standard determines that UAS shall have hazard, traffic, weather and anti-collision avoidance system.

DO-304 analyses potential circumstances when UAS share the same aerial space with manned aircraft. These situations involve communications between civil, military and commercial pilots and the UAV's operators. However, DO-304 does not give clear specified guidelines about which type of communications would be used to transmit the information between the civil aviation and the UAS.

A.2.2 RTCA DO-344 Operational and Functional Requirements & Safety Objectives for UAS

A close standard to DO-304 is the standard DO-344 [92] that includes issues concerning about the interaction between UAS and NAS. DO-344 refers to how the UAS should consider different factors to fly during daytime and night operations. Also, this standard contains the operational requirements for what UAS may share the airspace system with manned aircraft. All considerations related to UAS phases of flight, operational rules and regulations are explains in this standard.

A.3 Hardware and Software Standards

As mentioned before, the software development for airborne systems requires high conditions of safety in each process. At the same time, the hardware development processes for aerospace industry demand strict procedures that have as their first aim decreasing the probability of a system failure. To achieve this aim, the avionics manufacturers must certify their processes according to the standard DO-178, used to develop safety-critical software in airborne systems, and DO-254, used to develop Airborne Electronic hardware. The certification is evidence that an airborne company meets the safety criteria and therefore reduces the possibility that an unexpected failure happens during a flight.

The hardware and software development processes begin as independent processes. However, there is a point in which both processes must be integrated. At the same time, a safety assessment must independently be done for both the hardware and software prototypes during the development. When the software and hardware modules must include an integration phase, new safety assessment criterion is necessary to assess the entire development as shown in Figure A.3.

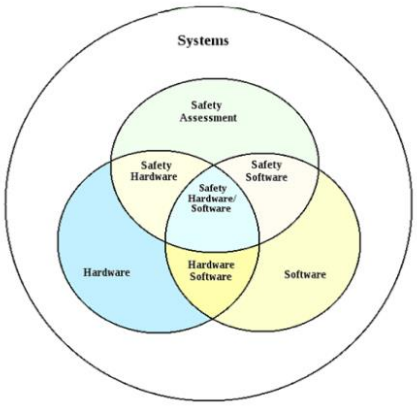


Figure A:3 Relationships among Airborne Systems, Safety Assessment, Hardware and Software Processes according RTCA DO-254 [91].

This section describes the software standard RTCA DO-178 "*Software Considerations for Airborne Systems and Equipment Certification*" and a series of standards that complement the software certification such as RTCA DO-332 "*Object-Oriented Technology and Related Techniques*", RTCA DO-330 "*Software Tool Qualification Considerations*". Also, some coding standards used by the software engineers to assure that the produced code be portable, modular and reliable are mentioned. After reviewing the software standards, is described the hardware standard DO-254 "*Design Assurance Guidance for Airborne Electronic Hardware*".

A.3.1 RTCA DO-178C. Software Considerations in Airborne Systems & Equipment Certification

The RTCA and EUROCAE decided to develop guidelines to increase the efficacy in the software industry for avionic systems emphasising the safety criteria. These considerations and instructions were promulgated in the airworthiness standard RTCA DO-178 and divide the software development into five processes. For each process the necessary activities, objectives, and inputs and outputs are outlined. This standard has evolved from its first version DO-178 to its recent version DO-178C. For the purposes of this chapter, the abbreviation DO-178 will be used, the comparison between the different versions is beyond scope of this appendix. To achieve certification, DO-178 does not specify a particular type of software, software modelling, programming language or software methodology. If an avionics manufacturer must achieve the certification for an equipment or airborne system, the company must demonstrate that process objectives were completed.

According to DO-178's suggested procedures a software design life cycle [93] should include a software planning process, a software development process and a certification process. The aim of dividing DO-178 into three processes is to establish clear guidelines about the communication and

cooperation among the different members of the development team to achieve the certification. The following will briefly analyse these processes.

A.3.1.1 Software Planning Process

This process defines the planning and organization of an avionics software project and its certification process [94]. The software planning process is related to the software life cycle, which means that the project's members must define the rules to know when a requirement has met the functional and safety tests or when a requirement has not met the safety controls. The way in which the software developers, testers and safety engineers provide feedback on the software modules to be redesigned, recoded and re-evaluated is planned in this process. Additionally, the software coding standards [95], programming languages, compilers, simulators and modelling methodologies [96] are defined in the planning process.

A.3.1.2 Software Development Process

This process defines the activities, input and outputs to generate a product or equipment for the avionics industry. This process is comprised of five processes which begin with the analysis of System Requirements and finish with the generation of a source code and the integration with a particular hardware. The software requirements, software design, software coding and integration are the processes defined in the software development.

A.3.1.3 Software Requirement Process

Figure A:4 Software Requirements Process

The inputs for this process include the analysis of the system requirements, the development of the system architecture and the type of hardware interfaces needed to implement the system. The system requirements do not include technical details but note the needs that the users require to interact with the system. Also, during this process the system requirements can be classified terms of functional and performance requirements, safety requirements and environmental requirements [97]. A methodology to identify and analyse requirements for safety-critical software using use-case modelling notation is Unified Modelling Language (UML) [98]. For instance, if an avionics manufacturer wishes to develop an Autopilot Flight System (AFS) for a commercial aircraft, one of the system requirements could be that the system has an altitude hold mode.

The output of this process is a document with the software requirements data that includes the information used by the system. For example, to develop an AFS, the essential data is the position of the servo motors, the aircraft altitude, the roll angle, the GPS location, and so on. This information will be used in the following process to design the software architecture and the low-level requirements.

A.3.1.4 Software Design Process

Figure A:5 Software Design Process

Using the data obtained in the software design process and the plan elaborated in the planning process, the development team select the software design standards. These standards define details about the source code, such as the naming conventions, description of methods, maximum number of nested calls and dynamic memory allocation. Furthermore, Real Time Systems features are chosen and include aspects like concurrency, global variables, interrupt driven programming and exception handling [90].

With the same three inputs the software architecture is implemented and the low-level requirements are defined. The software architecture defines if the project will use a Real Time Operative System (RTOS), Object-Oriented Programming (OOP) and Programming Patterns or only will use structured programming. The low-level requirements define a level of algorithm, the logic that will implement the system requirements or high level requirements.

A.3.1.5 Software Coding Process

Figure A:6 Software Coding Process

The software coding process uses the software architecture and the low-level requirements to write the source code according to programming language defined in the software planning process. However, to assure high reliability in the software coding process, the avionics industry must follow coding standards depending on the programming language that software engineers use.

A.3.1.6 Integration Process

Figure A:7 Integration Process

The integration process involves the source code and the software architecture to generate the executable file after compiling and linking the source code. It will be loaded into a hardware unit (Flash Memory, ROM), generating a software and hardware integration process. Additionally, a Parameter Data File (PDF) is created to load the necessary information into the hardware to run the executable file.

A.3.2 Levels of Software defined in DO-178

According to the level of impact that software failure could have in the flight safety conditions and the aircraft integrity, DO-178 classifies the failure conditions in five categories: catastrophic, hazardous/severe-major, major, minor, and No Effect [90]. For example, a program that presents a failure in one of the displays of the cockpit panel will be less risky than a failure in the altitude controls [98].

System Development Assurance Level	Failure Condition Classification
Level E	Failure has no impact
Level D	Failure impact is minor, noticeable but not critical to flight safety (e.g., passenger inconvenience)
Level C	Failure impact is major, safety-related but not severe (e.g, passenger discomfort but not injury)
Level B	Failure impact is severe (e.g., passenger injury)
Level A	Failure impact is catastrophic (e.g., aircraft crash)

Table A:1 Levels of critically of software components according to RTCA DO-178 [77].

A.3.3 Coding Standards and Programming Languages

The software coding process involves Software Coding Standards which offer rules and guidance for a specific programming language. DO-178 does not suggest a particular coding standard but describes features that the coding process must include. These features include the source code reliability, portability, maintainability, testability, reusability and readability. One of these standards is used for work in C, and C++ is the AV Coding Standard created by the Motor Industry Software Reliability

Association (MISRA) and specialized to use in vehicles systems [99]. A similar coding standard was developed by the Lockheed Martin Corporation named Joint Strike Air Vehicle (JSF++) [100] and the Jet Propulsion Laboratory (JPL) in 2006 sets Jet Propulsion Laboratory (JPL) rules which suggest ten rules to code safety critical applications [95].

There is a strong relation between the coding standards and the syntax of a programming language, as when a language has a more complicated syntax it may generate inappropriate programming practices or programmers may be prone to making mistakes. To avoid this type of situation, the coding standards help programmers to develop safety-critical applications by following rules that improve the coding process. For example, the MISRA-C (2004) proposes 122 mandatory rules that involve each aspect of C programming [99]. These rules prevent what can appear as usual problems in C programming, such as memory leaking, pre-processing and conversions. Moreover, the programming can turn very complicated and tedious for software engineers and hard to control for safety engineers. Many software companies offer tools that allow verifying if the programmers are writing the source code according the programming standards, which reduces development time.

The programming languages most often used to work in safety-critical systems for avionics products are ASM, C and C++, but alternative languages have been used in manned and unmanned aircraft. For example, interpreted languages such Java and Python have been an interesting alternative as their syntax is least complicated than C or C++ and their dynamic management memory is automatic. As a result, the applications can be developed in less time. However, these languages are interpreted and run on a virtual machine; as a consequence the computing processing is slower than the compiling languages.

One programming language widely used to program critical system in avionic equipment has been ADA. This language was created to meet with safety criteria. The common programming problem known as buffer overflow can be prevented using ADA [101]. Unlike C++, the ADA syntax is less complicated to use during the coding process. In addition, ADA is able to detect errors in compile-time instead of debugging and testing process [102].

Although DO178 does not mention a specific programming language companies such as Boeing develop their safety-critical embedded system using ADA, ASM, Jovial, C and C++. The National Aeronautics and Space Administration (NASA) has worked its projects using a fault tolerant architecture in ADA. The European Space Agency (ESA) requires ADA to work their mission critical systems and has been used in the projects Infrared Space Observatory (ISO), Solar and Heliospheric Observatory (SOHO) and Cassini Huygens [103].

A.3.4 RTCA DO-332 Object-Oriented Technology & Related Techniques

Although the Object-Oriented Technology (OOT) has been used during years in the software industry, the OOT in avionics software development had not been considered to design safety critical applications. However, the memory improvement and the processing speed in the microcontrollers and microprocessors have allowed OOT to be used in manned and unmanned aircraft. The RTCA published the specialised standard DO-332 [104] to provide guidelines about on the OOT and as supplement to standards DO-178C and DO-278A.

Some of the guidelines include characteristics like abstraction, encapsulation, polymorphism and overloading used in the Oriented Object Programming (OOP), also, DO-332 refers to exception management, dynamic memory management and virtualization [104]. The OOP brings many advantages, especially the advantage to reuse code, the application of modelling techniques [98] to implement requirements and software patterns [105] for software architecture design. These advantages allow software applications to have a modular design and decrease the development process time.

Notwithstanding, DO-332 warns about vulnerabilities in the use of OOP for avionics systems when the OOT can be poorly implemented. An inappropriate exception management may cause an unpredictable behaviour during run-time execution in the programs. The reusable software could become an inconvenient for critical systems because unnecessary functions or procedures increase the memory consumption and unused code [104].

A.3.5 RTCA DO-330 Software Tool Qualification Considerations

Hardware and software development require the use of software tools to create applications or products. In particular, for airborne systems there exists an exclusive standard which presents guidelines concerning software tool evaluation. DO-330 [106] is a standard that supports the software (DO-178) and hardware (DO-274) procedures but does not involve the software or hardware development process. The reason to publish this standard is that in the hardware and software development use different software tools like compilers, simulators, databases, version control software, emulators, etc. The RTCA suggests that these tools must certify a high level of reliability to be used in the software and hardware processes.

Similar to DO-178 and DO-254, the standard DO-330 suggests a tool life cycle processes comprised of a tool qualification planning process, a tool development processes and integral processes. At the same time, the integral process is comprised of a tool verification process, a tool configuration management process and a tool quality assurance process. Finally, a certification

Liaison process is the last step to audit an engineering tool which can be used for the development of critical systems. Certified software tools are essential what for a hardware or software system may be certified.

A.3.6 RTCA DO-254. Design Assurance Guidance for Airborne Electronic Hardware

Analogous to the standard RTCA DO-178 for airborne systems is the standard RTCA DO-254 for airborne electronic hardware. RTCA DO-254 [107] has many similarities to RTCA DO-178 regarding some of the processes to achieve certification. The standard RTCA DO-254 specifies the objectives, procedures, inputs and outputs for each hardware-related process according to safety requirements. RTCA DO-254 suggests a hardware design life cycle [93] that includes the hardware planning, design and support processes.

A.3.6.1 Hardware planning process

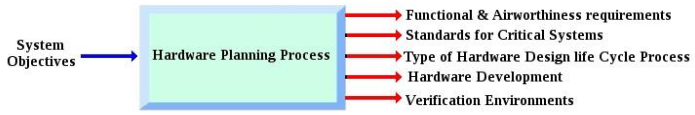


Figure A:8 Hardware Planning Process

The hardware planning process identifies the components, the design and verification methods and defines the methodology to control the development and maintenance with respect to hardware target [108]. The standard is not explicit about the types of hardware that should follow the standard guidance, but considers some examples. For instance, the production of complex hardware such as microprocessors, microcontrollers, digital signal processors, FPGS, graphic controllers, and circuit board assemblies must follow the DO-254 process to achieve airborne certification. Also, the interfaces SPI, I2C, RS232, analogue-digital converters should be audited using DO-254.

During the hardware planning are defined the functional and airworthiness requirements, are selected the methodologies to develop hardware. At the same time, verification environments are defined to evaluate the hardware target in different environmental, mechanical, magnetic and electrical conditions according to standard DO-160G [88].

A.3.6.2 Hardware Design Process

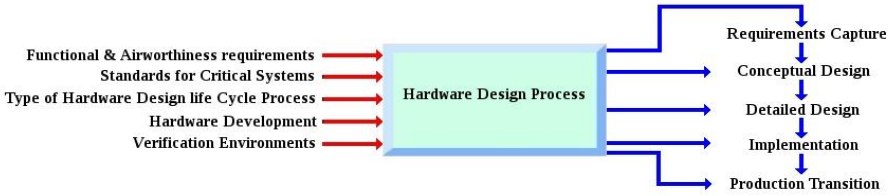


Figure A:9 Hardware Design Process

The hardware design defines the processes that should be followed to design and produce a hardware module, board or any digital device (Hardware item). This process is comprised of five internal processes which may have multiple iterations. The processes involved in the hardware design are: requirements capture, conceptual design, detailed design, implementation and production.

A.3.6.2.1 Requirements Capture Process

Figure A:10 Requirements Capture Process

The purpose of this process is to identify the hardware item requirements such as performance, architecture, encapsulation and functionality [91]. Also, this process must determine the safety requirements such as statistics analysis, to diagnose potential failures. During the requirements capture process additional requirements known as derived requirements, which add functionalities to system, may be found. Furthermore, the requirements traceability is established which, establishes the relation between the requirements, the developed code used to implement them and the test cases to verify if these requirements do not have failures.

A.3.6.2.2 Conceptual Design Process

Figure A:11 Conceptual Design Process

The conceptual design determines the design aims that will be implemented in the hardware system after the requirements are defined [108].

A.3.6.2.3 Detailed Design Process

Figure A:12 Detailed Design Process

In this process prototype are developed in a laboratory using the high level design concept and, the conceptual design data and the hardware item requirements are also defined. This prototype only could have some features of the system. Furthermore, a design assurance strategy is elaborated to validate the safety requirements according to Appendix B of DO-254 [109].

A.3.6.2.4 Implementation Process

Figure A:13 Implementation Process

During this process are detected errors or omissions and redirect to the appropriate process to be solved. The output of this process is the hardware item ready to be assembled.

A.3.6.2.5 Production Transition Process

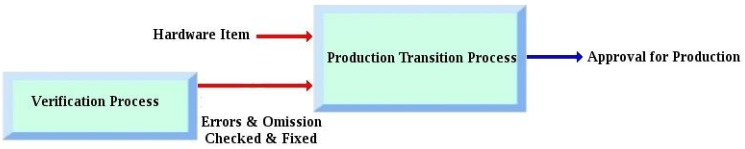


Figure A:14 Production Transition Process

This is the final process; where the product has been verified and has been approved indicating that the potential errors and omissions have been fixed. In this process the manufacturing requirements related to safety are determined and documented.

A.3.6.3 Validation and Verification Process

The validation process ensures that the hardware device has followed the right requirements to be developed and the verification process ensures that the hardware target in the design process meet the requirements [109].

A.3.6.4 Configuration Management Process

The main aim of this process is to manage control versions during the design process [108]. This control version is stored in a repository of data where it is possible to record, replicate or verify the changes effected during the hardware development.

A.3.6.5 The Certification Liaison

When the hardware manufacturer has finished the avionics hardware development and has completed the verification and validation processes, the manufacturer shall request an audit to of external organization authorised by the FAA. This organization will verify if all processes were met successfully to achieve the certification of the target hardware.

A.3.6.6 Levels of Hardware defined by DO-254

Similar to DO-178, DO-254 has classified the level of risk for a hardware item in case it could fail [107]. These levels establish safety criteria in the system requirements and increase precautions, improving the safety tests in hardware parts involved in the development. The hardware design assurance level definitions are in the following table.

System Development Assurance Level	Failure Condition Classification	Failure Condition Description
Level A	Catastrophic	Failure conditions that could cause an imminent interruption in safe flight conditions and landing.
Level B	Hazardous/Severe-Major	Failure conditions that could reduce the flight operations causing that pilots cannot steer the aircraft in suitable manner. As a consequence, the flight crew and passengers could suffer potential fatal injuries.
Level C	Major	Failure conditions that reduce the safety aircraft conditions and the manoeuvrability aircraft increasing the flight crew workload. These failures could cause changes in the flight itinerary and possible injuries to flight crew and passengers.
Level D	Minor	Failure conditions that would not engage the aircraft safety but would require that pilots take actions during the flight.
Level E	No Effect	Failure conditions that do not affect the operational capability of the aircraft or increase flight crew workload.

Table A:2 Levels of failure conditions according to DO-254 [91].

- **What considerations should an open source software and hardware autopilot project adopt for its use in the aviation industry?**

Unlike other type of software and hardware applications like small UAS platforms where safety and reliable standards are not imperative, the aircraft industry demands high safety levels in the software and hardware development processes. To achieve this aim the open-source projects related to avionics industry should follow a series of standards to ensure the safety requirements in the hardware and software processes.

The main difference between the certificated COTS hardware and software avionics products and open-source hardware and software projects is based on four considerations. The first is the analysis of safety requirements; the second is the analysis of functional requirements; the third is the processes of test to ensure the reliability of an airborne system and finally the elaboration of technical and user documentation.

With respect to the first consideration an open-source UAS AFS should elaborate safety requirements with high level of detail. The recommendations DO-332 and DO344 describe the aspects that an UAS must have to execute a safety flight considering functionalities such as flight recovery, flight guidance and traffic avoidance. Research, COTS and open-source projects focus their UAS AFS in one of three aspects but sometimes do not consider all aspects that DO-332 and DO-344. Therefore, an open-source UAS AFS would not be certified if any of these aspects was not considered.

Others standards that analyse the safety and functional requirements focusing on AFS are the standard DO-325. If an UAS AFS will need to be certificated, it should achieve the autopilot modes mention on this standard. However, DO-325 presents a main divergence that actually all open-source UAS AFS do not achieve. The divergence is that DO-325 refers to Human Machine Interfaces (HMI) on-board on the cockpit of a manned aircraft. It is understood that open-source UAS AFS do not consider the use of HMI because they do not require HMI on board on UAS. However, during this research demonstrated that open-source UAS AFS with high-performance hardware and software is able to interface HMI using CAN interface.

Avionics manufacturers must certificate their avionics modules in extreme environmental conditions such temperature, altitude, pressure and humidity. However, to get this certification for open-source UAS AFS would be complex because simulate strong atmospheric conditions in a laboratory demands high costs and time. Open-source projects often do not have the budget, the

physical infrastructure and the quantity of safety engineers and testers who test the modules to achieve high levels of verification.

With respect to the hardware and software processes open-source UAS AFS can develop or implement methodologies to increase the levels of safety during the hardware and software components. Often, the research, COTS and open-source projects are oriented to get technical aims in their products but they are not interested in the safety processes to achieve their objectives. DO-254 and DO178 consider that hardware and software processes dependent of safety criteria. To get this level of safety the functional requirement must have high level of detail. This rule implies to follow the guidelines proposed in DO-254 and DO-178 requires a high effort in time. The development of hardware and software for avionics products is tedious and complicated to create. This situation avoids that open-source UAS AFS could be certificated for both standards because the developers of open-source projects often collaborate with the projects part time as volunteers and do not spend time to follow rules about safety conditions and the hardware and software development.

Additionally, DO-254 and DO-178 emphasize that each hardware and software project must have an appropriate technical documentation and control version of source code. The lack of technical documentation is one of the main problems with open-source projects because the community of developers for open-source projects often contribute developing code but they do not write the technical documentation about the development. With respect to control of version of source code some open-source projects is enough organized but the quantity of developers contributing to develop applications does the control of versions cannot synchronized and generate errors of compilation.

Although, there are many reasons why open-source UAS AFS cannot be certified for avionics industry, it exists one example that open-source project can reach enough levels of quality and safety to be used in critical systems. GNU/Linux has been used for many companies to work in their avionics products. Many companies adopt the open-source projects to develop their own products due to this saves time and resources if it compares when an applications is developed since the beginning. For example, the company Lynx Software Technologies developed their Real Time applications using a modified Linux version calls Lynx-OS-178 which achieved with the standard DO-178. However, to achieve the level of certification DO-178 and DO-254 companies must evaluate and customize the open-source projects according their needs. This implies a positive aspect because many companies support the development of open-source projects releasing new improved versions which have been developed under the safety standards.

This Appendix introduced some of the most important standards to develop airborne applications that include unmanned and manned aircraft. All these mentioned standards provide guidelines to increase

the level of safety in the avionics industry or even in the UAV-related research projects. Although, to achieve level of certifications demands a huge effort in time, human and financial resources, but the advantages gained follow by following the procedures are huge, not only in terms of safety but also in terms of improving the hardware and software integration processes. Finally, considerations about open-source projects and the criteria that these should follow to be used in the avionics industry were analysed.

Appendix B: CAN Protocol

CAN protocol was developed by the German engineering company Bosch in 1985 to use in its automotive applications. The idea was to develop a specialized protocol to work in environments with high magnetic and electrical interference and mechanics. Furthermore, if new elements are added to a CAN network, additional modifications should not be necessary. This appendix describes the main concepts about CAN protocol especially those related to CAN Bus implementation.

Can protocol is developed in two layers, the physical and the data link layers [110].

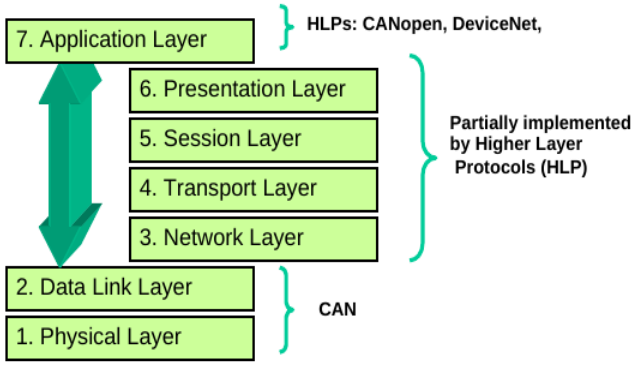


Figure B:1 ISO-OSI Reference model [110].

B.1 Physical Layer

The physical layer is the transmission medium and represents the electrical signal levels of a bit [110]. To transmit CAN messages on a CAN bus is necessary a cable with two wires, the first wire represents CAN high voltage (CAN_H) respect to ground around 2.5 and 4 volts and the second wire represents CAN low level (CAN_L) of voltage and varies from 2.5 to 1 volt [111]. The difference of voltages between both wires has a binary equivalent of "1" binary when the difference is 0 volts and "0" binary when the difference is 2 volts. It means that a contrary logic state exists between the CAN bus and the levels of voltage for a digital representation. As a result, a logic-high will be zero (recessive state) and a logic-low will be one (dominant state) on a CAN bus [112].

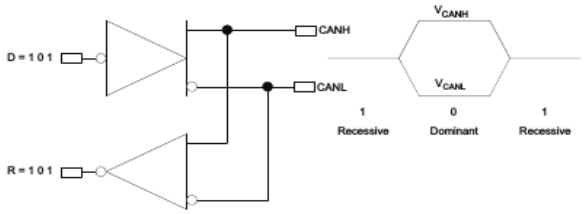


Figure B:2 Inverted logic for a CAN Bus [112, p.4].

B.2 Data Link Layer

According to [113] the data link layer is made of two layers; the Object layer which transmits the messages through CAN bus and filters messages received from the transfer layer, and the transfer layer that is considered the core of the CAN protocol and is in charged to control the logic related to framing, bus arbitration, error checking, error signalling, bit timing and fault confinement [113]. This layer analyses if the bus is available to send or accept a CAN message. The transfer layer manages four types of CAN frames: The data frame which has the data that a CAN node transmits, the remote frame which is used to request a data from a CAN node, the error frame which is emitted when a unit detects a bus error and the overload frame which is used to produce a delay between a data frame and remote frame [113].

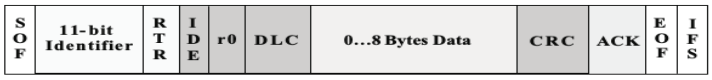


Figure B.3 Standard CAN: 11-Bit Identifier [112, p.3].

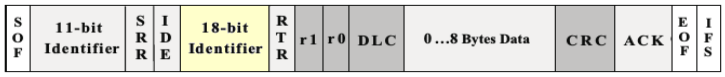


Figure B:4 Extended CAN: 29-Bit Identifier [112, p.4].

Bits of a standard CAN message

- SOF – Start of frame. This bit is used to start the transmission of a message.
- Identifier – Identifier of a CAN node over a CAN network.
- RTR – Remote Transmission Request. Bit dominant for remote frames and recessive for data frames.
- IDE – Dominant single identifier extension. A standard CAN identifier with no extension is transmitted.

- r0 – Reserved bit
- DLC – Data length code. Number of data bytes transmitted in the Data Field.
- Data – Data transmitted in a frame.
- CRC – Cyclic Redundancy Check. Contains the checksum.
- ACK – This 2-bit field is used to confirm if a message is error-free or if the message must be repeated by the sending node.
- EOF – End-of-frame (EOF), 7-bit field "marks the end of the CAN frame.
- IFS – 7-bit interframe space (IFS)

B.3 CAN Node

A CAN node is an element that transmits and receives messages on a CAN Network and is comprised of a digital part (microcontroller) and a physical part (transceiver). The microcontroller sends and accepts binary information using pins TxD and RxD, at the same time these pins are connected to pins RXD and TXD of the physical part (transceiver). The transceiver produces the differential level of voltages used in a CAN Bus (CAN_H, CAN_L) using pins CANH and CANL [110]. A CAN node can be a sensor, a servo motor or an interface interconnected to a CAN bus. Each node has a assigned identifier which can be of 11 bits when CAN version is 2.0A or can be of 29 bits when CAN version is 2.0B [113].

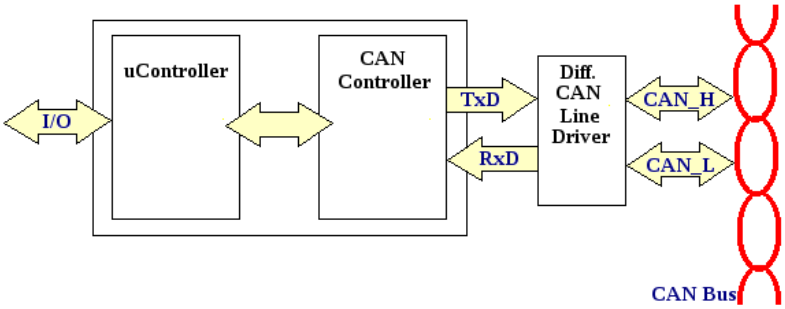


Figure B:5 Example of CAN Node [114].

B.4 CAN Bus

The CAN protocol allow each node to communicate with other nodes over a Network. The communication is multicasting, it means that all messages over a CAN network are listened by all nodes. However, each node decides what messages must be filtered depending on CAN identifier and data [113]. CAN protocol is highly flexible and allows that one or more nodes can be added without modifying the software and hardware [113]. In a CAN Network there is no central node which distributes CAN frames over the network, therefore a linear bus topology can easily be implemented (Figure B.6) [114]. The priority of a CAN message depends on the identifier value where a lower binary value has the higher priority to transmit over CAN bus.

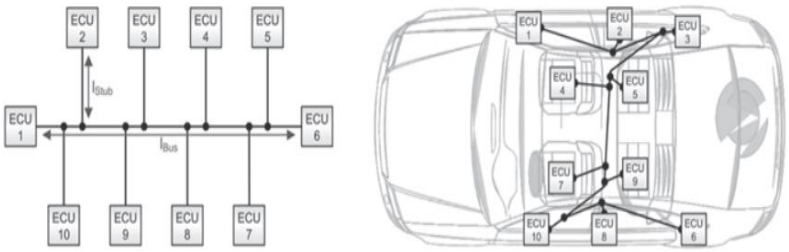


Figure B.6: Diagram of a CAN linear bus topology [114].

B.4.1 Bus Access (Arbitration)

A remarkable characteristic of CAN protocol is the arbitration mechanism known as Carrier Sense Multiple Access with Collision Detection and Arbitration protocol (CSMA/CD + AMP) [115]. This type of arbitration is used when two or more nodes try sending CAN frames at the same time. To avoid conflicts when various nodes transmit simultaneously CAN messages a bitwise arbitration exists where a lowest identifier node will have the highest priority to transmit on CAN Bus [112]. The following example explains the CAN bus arbitration.

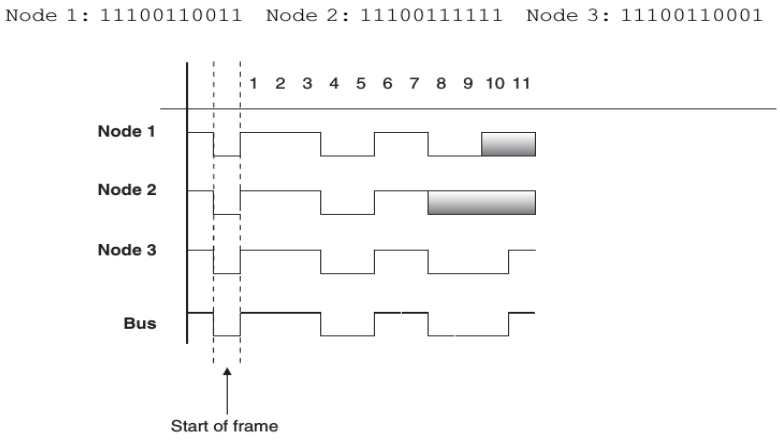


Figure B:7: Example CAN bus arbitration [116].

Figure B.7 shows three nodes which start simultaneously to transmit after all nodes have waited that CAN Bus was free. First, the three nodes synchronise their transmissions using the bit start of frame (SOF) and transmit their node identifiers using CAN standard version 2.0A. The first seven bits for each node identifier are equal, however, after on 8th bit the node 2 transmits one putting its state in recessive. This means that node 2 stops its transmission and changes to reception state. The nodes 1 and 3 continuing transmitting until 10th bit until node 1 transmits 1 changing its state to recessive. As a result, node 1 stops its transmission and changes to a reception state. Finally, the node 3 wins the arbitration and changes to dominant state where can send its CAN frame [116].

Appendix C: **H-Bridge**

The Appendix C shows an overview about H-Bridge driver functionality. A standard H-Bridge has four MOSFET transistors which acted as switches working in couples to invert the servo motor polarity as shown in Figure C.1 [117]. When MOSFETS S1 and S4 are activated the current flows from positive voltage through the servo until ground supply producing that servo goes forward (Figure C.2.a). If the sense of position in servo must change, MOSFETS S2 and S3 are activated whereas MOSFETS S1 and S4 are disabled (Figure C.2.b) producing that servo goes in reverse direction [117]. When all four MOSFETS are switched off, the servo motor changes a neutral condition (Figure C.2.c).

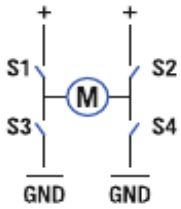


Figure C:1 Four-Switch H-bridge [117].

Using servos with high or medium torque and working with external load can be necessary to break them before changing its sense. The H-Bridge brings two possibilities to stop a servo plugging two terminals at the same voltage. The first option is to close the transistors S1 and S2 which produces a positive electric brake (Figure C.2.d) and the second option is to close the transistors S3 and S4 which produces a negative electric brake (Figure C.2.e).

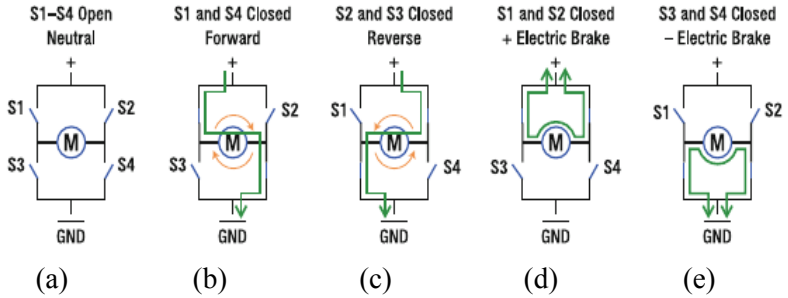


Figure C:2 H-Bridge Operation [117].

Appendix D: **Reconfigurable Autopilot Flight System**

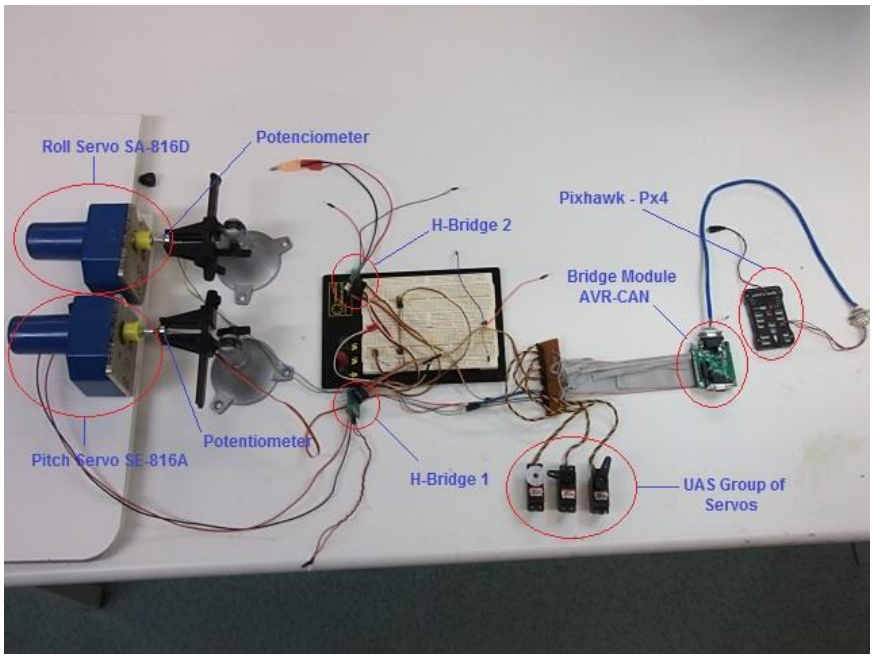
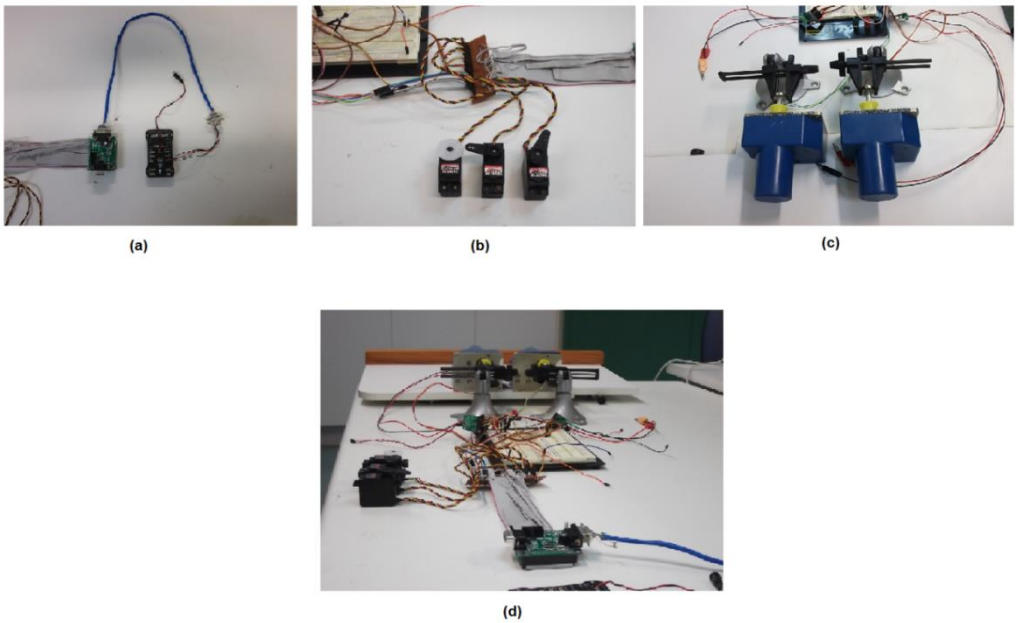


Figure D:1 Reconfigurable Autopilot Fligth System Components



(a) Pixhawk Autopilot (Right) and AVR-CAN Board (Left) (b) UAS Servos
(c) Cessna Servos (d) Front View RAFS

Figure D:2 Reconfigurable Autopilot Flight System

Appendix E: **Px4 Pixhawk Schematic I/O**

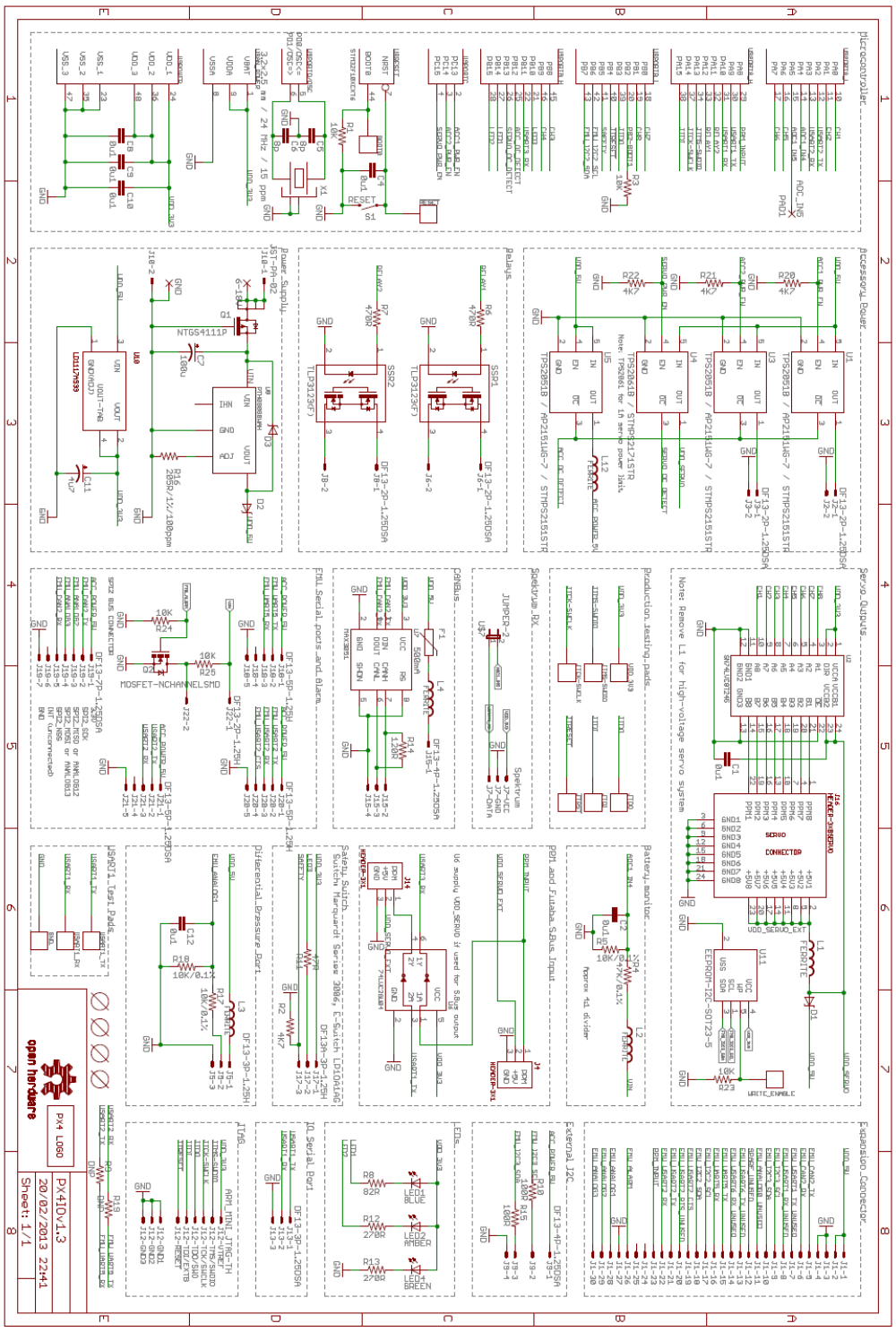


Figure E:1 Pixhawk* UAS Autopilot Schematic [51].

** Pixhawk Autopilot is an open-source hardware and software project under the non-copyleft free software license BSD (Berkeley Software Distribution).*

Appendix F: **AVR-CAN Board Schematic**

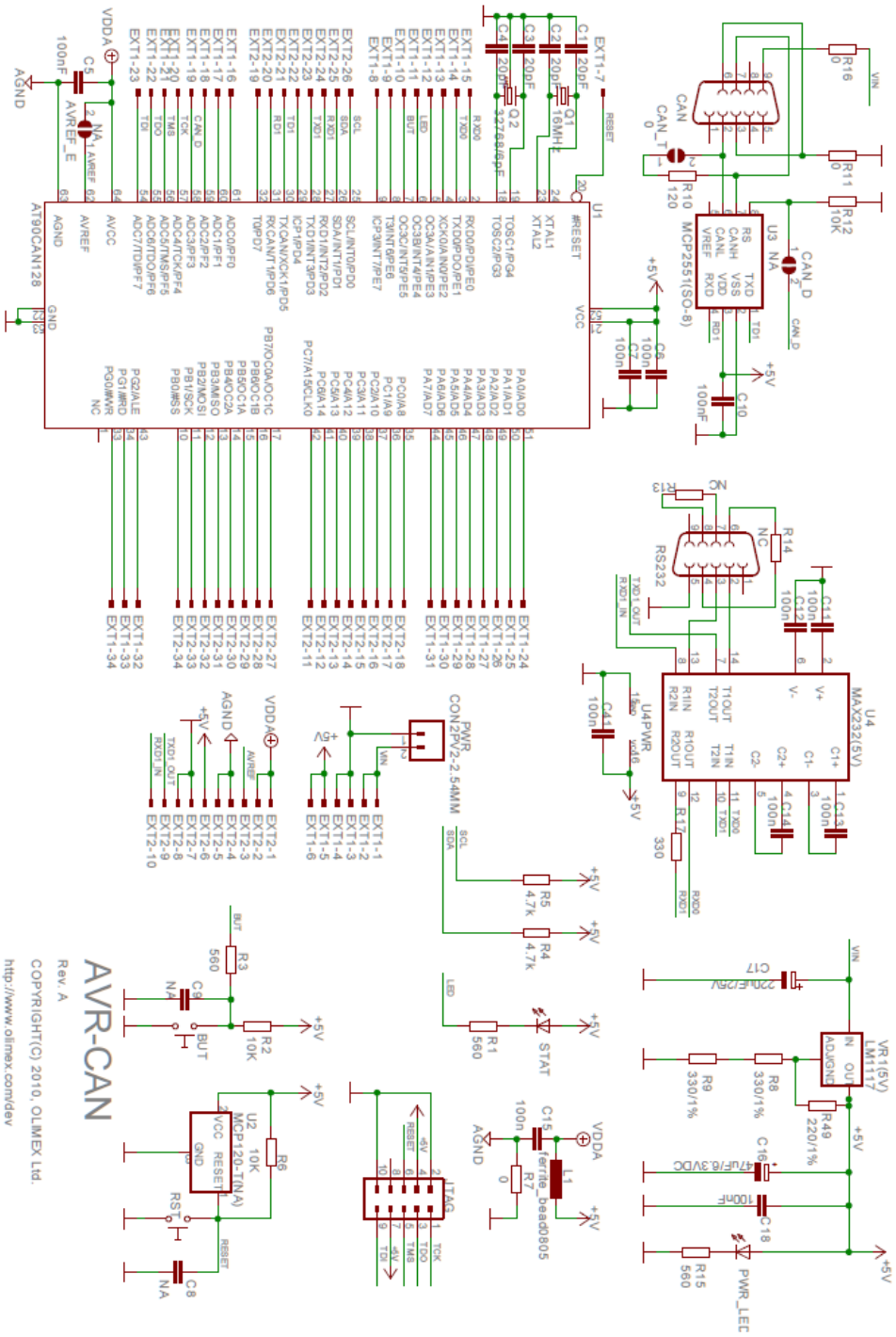


Figure F:1 AVR-CAN* Board Schematic [63].

** Copyright(c) 2011, OLIMEX Ltd, All rights reserved.*

Appendix G: **Control H-Bridge 33926**

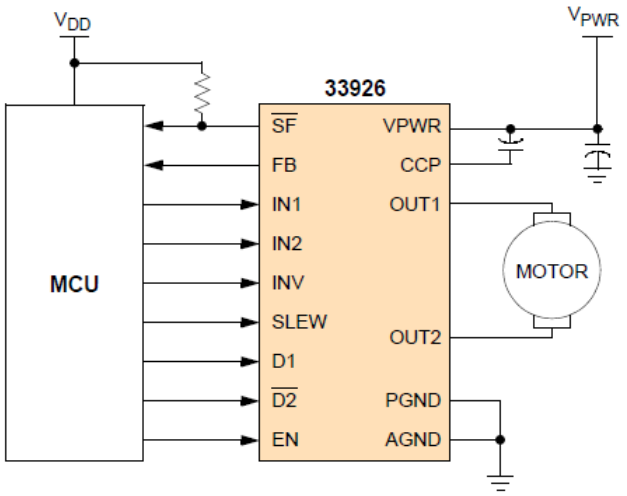


Figure H:1 H-Bridge driver 33926* Wiring diagram with a microcontroller and servo [119].

Device State	Input Conditions					Status	Outputs	
	EN	D1	$\overline{D2}$	IN1	IN2	$\overline{SF}$	OUT1	OUT2
Forward	H	L	H	H	L	H	H	L
Reverse	H	L	H	L	H	H	L	H
Free Wheeling Low	H	L	H	L	L	H	L	L
Free Wheeling High	H	L	H	H	H	H	H	H
Disable 1 (D1)	H	H	X	X	X	L	Z	Z
Disable 2 ($\overline{D2}$)	H	X	L	X	X	L	Z	Z
IN1 Disconnected	H	L	H	Z	X	H	H	X
IN2 Disconnected	H	L	H	X	Z	H	X	H
D1 Disconnected	H	Z	X	X	X	L	Z	Z
$\overline{D2}$ Disconnected	H	X	Z	X	X	L	Z	Z
Under-voltage Lockout (30)	H	X	X	X	X	L	Z	Z
Over-temperature (31)	H	X	X	X	X	L	Z	Z
Short-circuit (31)	H	X	X	X	X	L	Z	Z
Sleep Mode EN	L	X	X	X	X	H	Z	Z
EN Disconnected	Z	X	X	X	X	H	Z	Z

Table H:1 Truth Table* with the logic commands for H-Driver 33926 [119].

Notations: L = LOW, H= HIGH. X=HIGH or LOW, and Z = High Impedance.

References

[1] D. Vos. (2009). *Five steps to facilitating the convergence of manned and unmanned aviation* [Online]. Available: http://uvs-international.org /phocadownload/03_5bc_Relevant_Information/Content_Five-steps-to-facilitating-the-convergence-of-manned-and-unmanned-aviation_Rockwell-Collins.pdf

[2] R. Sterling (2013). *"On target: F-16 flies with an empty cockpit"*. Boeing [Online]. Available: http://www.boeing.com/boeing/Features/2013/09/bds_qf16_09_23_13.page

[3] C. Cuttita. (2013, Nov 11). *Brains behind the QF-16 spearhead AF aerial target drone program* [Online]. Available: http://www.aerotechnews.com/news/2013/11/01/brains-behind-the-qf-16-spearhead-af-aerial-target-drone-program/

[4] A. Gatti. (1997, October). *From manned and unmanned: a viable alternative to the scrapyard* [Online]. Available: http://ftp.rta.nato.int/public/PubFulltext/AGARD/ CP/AGARD-CP-594/05SE1-02.pdf

[5] B. Waldman. (2010, Nov). *ARINC429 specification tutorial AIM GmbH* [Online]. Available: http://www.aim-online.com/pdf/OVIEW429.PDF

[6] N. Rieckman, "ARINC 629 data bud physical layer technology," *Microprocessors and Microsystems*, vol. 21, pp. 13-90, July 1997.

[7] J. Kluser, (2012, April). *CAN-based protocols in avionics*. [Online]. Available: https://www.vector.com/portal/medien/cmc/application_notes/AN-ION-1-0104_CAN-based_protocols_in_Avionics.pdf

[8] General Standardization of CAN (Controller Area Network) Bus Protocol for Airborne Use, ARINC Specification 825-2, 2011.

[9] CAN Open protocol. [Online]. Available: http://www.can-cia.org/index.php?id=specifications&no_cache=1

[10] M. Stock (2006, Jan. 12), "*CANaerospace interface specification for airborne CAN Applications V 1.7*". Stock Flight Systems [Online]. Available: http://www.stockflightsystems.com/tl_files/downloads/canaerospace/canas_17.pdf

[11] K. Randall, "ARINC 429 Digital data communications for commercial aircraft," *Journal Guidance, Control, and Dynamics*, vol. 6, no. 2, pp. 120-123, March 1983.

[12] L. Parrilla et al., "Design of a middleware interface for ARINC 429 data bus," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, pp. 1136-1149, April 2012.

[13] R. Knueppel, *Standardization of CAN networks for airborne use through ARINC 825.* Airbus Operations GmbH [Online]. Available: http://www.can-cia.org/fileadmin/cia/files/icc/13/knueppel.pdf

[14] J. Park and S. Mackay, *Practical Data Acquisition for Instrumentation and Control Systems*. Newnes, 2003.

[15] D. Johnson, "Interfaces using general purpose digital instrumentation," *IEEE Instrumentation & Measurement Magazine*, vol. 13, pp. 8-13, August 2010.

[16] R. Collinson. (2011). *Introduction to avionics systems.* (3rd edition). [Online]. Available: http://link.springer.com.ezp01.library.qut.edu.au/book/10.1007%2F978-94-007-0708-5

[17] C. M. Ananda, "General aviation light transport aircraft avionics: Integration and systems tests," in *Digital Avionics Systems Conference, 2007. DASC '07. IEEE/AIAA 26th*, Bangalore, India, 2007, pp. 2.A.3.

[18] MGL Avionics. (2011, March 10). *EFIS Integrated autopilot*. [Online]. Available: http://www.mglavionics.com/Autopilot.pdf

[19] GRT Avionics. (2013, Nov 22). *GRT autopilot installation.* [Online]. Available: http://www.grtavionics.com/GRT%20Autopilot%20Installation-Setup.pdf

[20] Trutrak Flight Systems. (2009, Sept 14). *Installation manual for EFIS autopilots Rev E.* [Online]. Available: www.trutrakflightsystems.com

[21] Garmin Ltd. (2004, Nov). *G1000 guide for designated pilot examiners and certified flight instructors.* [Online]. Available: http://www.capmembers.com/media/cms/G1000Instructors_Guide_C67700AAF429E.pdf

[22] Bendix/King Company. (1998, Jun). *KAP 140. Pilot's guide.* [Online]. Available: http://www.n612sp.com/KAP%20140%20AUTOPILOT.pdf

[23] Garmin Ltd. (2013, Feb). *System maintenance manual.diamond DA 40 & DA 40 F including GFC 700 or KAP 140 AFCS.* [Online]. Available: http://static.garmincdn.com/pumac/190-00545-01.pdf

[24] MGL Avionics. (2010, Oct 20). *Servo specifications and installation manual*. [Online]. Available: http://www.mglavionics.com/Servo_2010_10_20_v19.pdf

[25] MGL Avionics. *MGL Autopilot servo interface protocols.* [Online]. Available: http://www.mglavionics.co.za/Docs/Servo%20communications%20protocol.pdf

[26] C. Ananda. *Civil aircraft advanced avionics architectures.* [Online]. Available: http://nal-ir.nal.res.in/4204/1/9._Civil_Aircraft_advacned_avionics_architectures_-_Shri._C.M._Ananda.pdf

[27] R. Hornish. "777 Autopilot flight director system," in *Digital Avionics Systems Conference, 1994. 13th DASC, AIAA/IEEE,* Phoenix, AZ, USA, Oct 30- Nov 03, 1994, pp. 151-156.

[28] D. Jung et al., "Design and development of a low-cost test-bed for undergraduate education in UAVs," in *Decision and Control, 2005 and 2005 European Control Conference, CDC-ECC'05. 44th IEEE Conference on,* Seville, Spain, December 12-15, 2005, pp. 2739-2744.

[29] D. Lara et al., "Onboard system for flight control of a small UAV," in *World Automation Congress (WAC),* Puerto Vallarta, Mexico, June 24-28, 2012, pp. 1-6.

[30] D, Maranhao and P. Alsina, "Project of a hardware and software architecture for an unmanned aerial vehicle," in *Robotics Symposium (LARS), 2009 6th Latin American,* Valparaiso, Chile, Oct 29-30, 2009, pp. 1-6.

[31] S, Rogers et al., "Adaptive autopilot system for small fixed wing UAVs," in *Aerotech*

Congress and Exhibition, Grapevine, Texas, USA, Oct 3-6, 2005, Sae technical paper series. doi: 10.4271/2005-01-3355.

[32] E. Atkins et al., "A reconfigurable flight management system for small-scale unmanned air systems," in *AIAA Infotech Aerospace Conference*, Seattle, Washington, USA, April 6-9, 2009.

[33] V. Tretyakov and H. Surmann, "Hardware architecture of a four-rotor UAV for USAR/WSAR scenarios," in *International Conference on Simulation, Modelling and Programming for Autonomous Robots*, Venice, Italy, November 3-4, 2008, pp. 434-443.

[34] M. Ax et al., "Architecture of an autonomous mini unmanned aerial vehicle based on a commercial platform," in *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, Munich, Germany, June 7-9, 2010, pp. 1-6.

[35] I. Lizarraga et al. (2009, August) *Low cost rapidly reconfigurable UAV autopilot for research and development of guidance, navigation and control algorithms.* [Online]. Available: http://mechatronics.ece.usu.edu/uav%2Bwater/MESA09-SUAVTA/DETC2009-86547.pdf

[36] T. Haifeng and D. Xiaojing, "The design of small UAV autopilot hardware system based on DSP," in *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on*, Changsha, China, May 11-12, 2010, pp. 780-783.

[37] M Cunxiao and F. Jiancheng, "The design and implementation of micro autopilot system for low-altitude mapping of MAV," in *Mechanic Automation and Control Engineering (MACE), 2011, Second International Conference on*, Hohhot, China, July 15-17, 2011, pp. 1744-1748.

[38] H. Christophersen et al. (2004, September). *Small adaptive flight control systems for UAVs using FPGA/DSP technology*. [Online]. Available: http://soliton.ae.gatech.edu/people/ejohnson/uav2004_fcs20.pdf.

[39] W. Alvis et al. "Fpga based flexible autopilot platform for unmanned systems," in *Mediterranean Conference on Control and Automation, 2007, IEEE*. doi: 10.1109/MED.2007.4433818

[40] P. Scherz et al., "Embedded Sensor Fusion System for Unmanned Vehicle Navigation," in *Mechtronic and Embedded Systems and Applications, 2008. MESA 2008. IEEE/ASME*

International Conference on, Beijing, China, Oct 12-15, 2008, pp. 192-197.

[41] V. Ragavan et al., "A reconfigurable FPGA framework for data fusion in UAV's," in *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, Coimbatore, India, Dec 9-11, 2009, pp. 1626-1631.

[42] R. H. Klenke, "A UAV-Based computer engineering capstone senior design project," in *Microelectronic Systems Education, 2005. (MSE '05). Proceedings. 2005 IEEE International Conference on*, June 12-14, 2005, pp. 111-112.

[43] O. Spinka et al., "Reconfigurable control system for small UAVs", *Industrial Electronics, IEEE Transactions on*, vol. 58, pp. 880-889, February. 2011.

[44] A. Catena et al., "An architecture for automatic tuning of the navigation system of unmanned aerial vehicles," in *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, Atlanta, Georgia, USA, May 28-31, 2013, pp. 592-598.

[45] D. Zhicheng et al., "A Reconfigurable Flight Control System Architecture for Small Unmanned Aerial Vehicles," *Systems Conference (SysCon), 2012 IEEE International*, Vancouver, BC, Canada, March 19-22, 2012, pp. 1-4.

[46] E. Santamaria et al., "Increasing UAV capabilities through autopilot and flight plan abstraction," in *IEEE/AIAA 26th Digital Avionics Systems Conference, 2007. DASC '07*, Dallas, TX, USA, Oct 21-25, 2007, pp. 5.B.5-1 – 5.B.5-10.

[47] P. Doherty et al. (2004). *A distributed architecture for autonomous unmanned aerial vehicle experimentation* [Online]. Available: http://www.ida.liu.se/~frehe/publications/dars2004.pdf

[48] Piccolo Autopilots. UTC Aerospace Systems [Online]. Available: http://www.cloudcaptech.com/piccolo_system.shtm

[49] Kestrel Flight Systems. (2014). Lockheed Martin Corporation [Online]. Available: http://www.lockheedmartin.com.au/us/products/procerus/kestrel.html

[50] Paparazzi project. (2014). [Online]. Available: http://wiki.paparazziuav.org/wiki/Main_Page

[51] PX4 Pixhawk Autopilot. Computer Vision and Geometry Lab [Online]. Available: https://pixhawk.org/choice

[52] 3D Robotics Inc. Lab. [Online]. Available: http://ardupilot.com/

[53] OpenPilot project. (2010). [Online]. Available: http://wiki.openpilot.org/pages/viewpage.action?pageId=21856869

[54] Omni Instruments. (2014). Microbot Autopilot [Online]. Available: http://www.omniinstruments.com.au/products/product/moredetails/microbot.id113.html

[55] PC/104 Consortium. [Online]. Available: http://www.pc104.org/specifications.php

[56] STM32F427xx. (2014). STMicroelectronics [Online]. Available: http://www.st.com/web/en/resource/technical/document/datasheet/DM00071990.pdf

[57] MAX3051. Maxim Integrated Products, Inc [Online]. Available: http://datasheets.maximintegrated.com/en/ds/MAX3051.pdf.

[58] J. Bard, "The POSIX operating system," in *Software defined radio: the software communications architecture,* Chichester, UK: Wiley, 2007, pp. 253.

[59] E. Wolfgang et al (ed.), "Hardware-dependent Software. Principles and Practice", Springer ISBN: 978-1-4020-9435-4, 2009.

[60] Toolchains. (2014) [Online]. Available: http://elinux.org/Toolchains

[61] GNU Tools for ARM Embedded Processors. [Online]. Available: https://launchpad.net/gcc-arm-embedded.

[62] S. Corrigan, "Controller area network physical layer requirements," Texas Instruments, Dallas, Texas. Rep. SLLA270, Jan. 2008.

[63] AVR-CAN development board. Olimex [Online]. Available: https://www.olimex.com/Products/AVR/Development/AVR-CAN/resources/AVR-CAN.pdf

[64] 8-bit AVR Microcontroller with 32k/64K/128K Bytes of ISP Flash and CAN Controller.

[Online], Available: http://www.atmel.com/Images/doc7679.pdf

[65] Atmel Extension for Atmel studio 6 end user license agreement. [Online]. Available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&cad=rja&uact=8&ved=0CCQQFjAB&url=https%3A%2F%2Fgallery.atmel.com%2FBlobs%2FDownload%2F4c45dceb-e1b7-47a6-bb8a-ab1f8cfe0bdf&ei=CK9YVIPJFs3c8AXLu4DYBg&usg=AFQjCNElcA4tqFTnxpMkohtXVRobLaNgKw&bvm=bv.78677474,d.dGc

[66] Atmel Studio: Release 6.0. 8/32-bits Atmel Microcontrollers [Online]. Available: http://www.atmel.com/images/as6installer-6.0.1843-readme.pdf

[67] GNU General Public License. [Online]. Available: http://www.gnu.org/copyleft/gpl.html

[68] AT90CAN128 Support for Arduino. [Online]. Available: https://github.com/lincomatic/AT90CAN

[69] Arduino. [Online]. Available: http://www.arduino.cc/

[70] AVR Libc. [Online]. Available http://www.nongnu.org/avr-libc/

[71] AT90CAN128 ported to Arduino. [Online]. Available: http://forum.arduino.cc/index.php/topic,8446.0.html

[72] Olimex ATMEGA128 Header board. [Online]. Available: https://www.olimex.com/Products/AVR/Header/AVR-H128-C/

[73] HS-645MG. OpenServo.org. [Online]. Available: http://www.openservo.com/Servo_HiTec_HS-645MG

[74] HS-945MG. Robotzone [Online]. Available: http://www.servocity.com/html/hs-985mg_super_torque.html

[75] *Maintenance Manual FCS-810*, Bendix Avionics Division, 1976.

[76] *FCS-810 Flight Control System Installation Manual*, Bendix Avionics Division, 1970.

[77] *Software considerations in airborne systems and equipment certification,* RTCA DO-178C, 2011.

[78] *Guidelines for the use of the C Language In Vehicle Based Software,* MISRA, 1998.

[79] The C Preprocessor. (2001). [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc-2.95.3/cpp_1.html#SEC29

[80] *STM32F42xxx and STM32F43xxx ARM-based 32-bit MCUs Reference manual*, STMicroelectronics, 2014.

[81] Hitec. General Servo Information. [Online]. Available: http://hitecrcd.com/files/Servomanual.pdf

[82] T. Mansour, Ed. (2011, April 19). *PID control, implementation and tuning*. [Online]. Available: http://www.intechopen.com/books/pid-control-implementation-and-tuning

[83] S. Sung. (2009, July). *Process Identification and PID Control.* [Online]. Available: http://site.ebrary.com/lib/qut/detail.action?docID=10325864

[84] K. Altaf et al., "Design, implementation and real-time digital control of a cart-mounted inverted pendulum using Atmel AVR Microcontroller," in *International Conference on Signal Processing, Robotics and Automation 6th Proceedings of the WSEAS,* February 16–19, 2007.

[85] G. Liping., "Implementation of digital PID controllers for DC-DC converters using digital signal processors", Electrical & Information Engineering Technology, IEEE, vol. no, pp 306-311, May 2007.

[86] Lawicel CANUSB Manual. [Online]. Available: www.canusb.com

[87] *RTCA Minimum Operation Performance Standards (MOPS) for Automatic Flight Guidance and Control Systems and Equipment,* RTCA Standard DO-325, 2011.

[88] *RTCA Environmental Conditions and Test Procedures for Airborne Equipment*, RTCA Standard DO-160G, 2010.

[89] *RTCA Guidance Material and Considerations for Unmanned Aircraft Systems,* RTCA Standard DO-304, 2007.

[90] *RTCA Software Considerations in Airborne Systems and Equipment Certification,* RTCA Standard DO-178C, 2011.

[91] *RTCA Design Assurance Guidance for Airborne Electronic Hardware,* RTCA Standard DO-254, 2000.

[92] *RTCA Operational and Functional Requirements and Safety Objectivens (OFRSO) for Unmanned Aircraft System (UAS),* RTCA Standard DO-344, 2013.

[93] N. Priggouris et al., "The System Design Life Cycle," in *Cost-efficient Methods and Processes for Safety-relevant Embedded System,* Spring-Verlag ISBN: 9783709117255, 2013.

[94] P. B. Carpenter. (October, 1999). *Verification of requirements for safety-critical software.* [Online]. Available: http://delivery.acm.org/10.1145/320000/319299/p23-carpenter.pdf?ip=131.181.251.12&id=319299&acc=ACTIVE%20SERVICE&key=65D80644F295BC0D%2ECE8691788DF0BE02%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=461297627&CFTOKEN=25786282&__acm__=1417955728_01e8de61097dd69b1d41b36e4cf3daf1

[95] P. Anderson., "Coding Standards for High-Confidence Embedded Systems," in *Military Communications Conference, MICOM, 2008 IEEE*, San Diego, CA, USA, November 16-18, 2008, pp. 1- 7.

[96] J. Rumbaugh et al., *The unified modelling language reference manual*. Boston, MA: Addison-Wesley. 2005.

[97] P. Wang and Y. Tian, "Study on formal methods application airborne electronic hardware design," in Software Engineering (WCSE), 2010 Second World Congress on, Wuham, China, December 19 – 20, 2004.

[98] G. Zoughbi et al., "Modeling safety and airworthiness (RTCA DO-178B) information: conceptual model and UML profile," in *Software and Systems Modelling,* Springer-Verlag, 2011, pp. 337-367.

[99] *Guidelines for the Use of the C Language*, The motor Industry Reliability Association MISRA, 1998.

[100] *Joint Strike Air Vehicle C++ Coding Standards For the System Development and Demonstration Program*, Lockheed Martin, 2005.

[101] Ada 2012 Language Rationale Published. (2013, Nov 12). [Online]. Available: http://www.businesswire.com/news/home/20131112005610/en/Ada-2012-Language-Rationale-Published#.VIR27Nc7fQg.

[102] ADAcore. (2012). [Online]. Available: http://www.ada2012.org/

[103] C. Johnson. *Safety-Critical software development DO-178B.* [Online]. Available: http://www.dcs.gla.ac.uk/~johnson.

[104] *RTCA Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A,* RTCA Standard DO-332, 2011.

[105] B. P. Douglass. (2011). *Design patterns for embedded systems in C*. [Online]. Available: http://www.sciencedirect.com.ezp01.library.qut.edu.au/science/book/9781856177078

[106] *RTCA Software Tool Qualification Considerations,* RTCA Standard DO-330, 2011.

[107] *RTCA Design Assurance Guidance for Airborne Electronic Hardware,* RTCA Standard DO-254, 2000.

[108] P. S. Miner et al. *A case-study application of RTCA DO-254: Design assurance guidance for airborne electronic hardware.* [Online]. Available: https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/case_study_do-254.pdf

[109] M. Lange. (2013, Sep 3). *DO-254 Basics part1: Development history and invocation.*[Online]. Available: https://www.youtube.com/watch?v=prt9Ribxe4s

[110] AVR32129: Using the 32-bit AVR UC3 CANIF. (2010). Atmel Corporation. [Online]. Available: http://www.atmel.com/Images/doc32152.pdf

[111] CAN: Primer creating your own network. Keil. [Online]. Available:

http://www.keil.com/download/files/canprimer_v2.pdf

[112] Introduction to the Controller Area Network (CAN). (2008). Texas Instruments. [Online]. Available: http://www.ti.com/lit/an/sloa101a/sloa101a.pdf

[113] Bosh CAN Specification. (1991). Robert Bosh GmbH [Online]. Available: http://esd.cs.ucr.edu/webres/can20.pdf

[114] W. Lawrenz (ed.), "CAN System Engineering From Theory to Practical Applications", Springer ISBN: 978-1-4471-5612-3, 2013.

[115] H, Choi et al., "Multiple Access with Collision Resolution", IEEE COMMUNICATIONS LETTERS, vol. 17, pp. 1284 – 1287, February 2013.

[116] D. Ibrahim., "Advanced PIC microcontroller projects in C: from USB to ZIGBEE with the PIC", Elsevier Ltda ISBN: 978-0-7506-8611-2. 2008, 2008.

[117] J. Warren et al., "Arduino Robotics", Springer Science ISBN: 978-1-4302-3183-7, 2011.

[118] Y. Luo and X. Chen, "Data acquisition and communication system of miniature UAV based on DSP," in *Control Engineering and Communication Technology (ICCECT), 2012 International Conference on*, Dec 7-9, 2012, pp. 736-740.

[119] MC33926 Motor Driver Carrier. (2014). Pololu Corporation. [Online]. Available: http://www.pololu.com/product/1212

[120] *Supplemental airplane maintenance manual DA 42 NG retrofit installation of austro engine E4-B,* Diamond Aircraft., Wiener, Austria, 2009.